JÉSSICA RENATA NOGUEIRA

UMA AVALIAÇÃO DE SEQUÊNCIAS DE INSERÇÃO EM ALGORITMOS INCREMENTAIS PARA A TESSELAÇÃO DE DELAUNAY

LAVRAS - MG

2015

JÉSSICA RENATA NOGUEIRA

UMA AVALIAÇÃO DE SEQUÊNCIAS DE INSERÇÃO EM ALGORITMOS INCREMENTAIS PARA A TESSELAÇÃO DE DELAUNAY

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Inteligência Computacional e Processamento Digital de Imagens, para a obtenção do título de Mestre.

Orientador

Dr. Sanderson Lincohn Gonzaga de Oliveira

LAVRAS - MG

2015

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo próprio autor.

Nogueira, Jéssica Renata.
Uma avaliação de sequências de inserção em algoritmos incrementais para a tesselação de Delaunay/ Jéssica Renata Nogueira. – Lavras : UFLA, 2015.
119 p. : il.

Dissertação (mestrado acadêmico) – Universidade Federal de Lavras, 2015.
Orientador: Sanderson Lincohn Gonzaga de Oliveira.
Bibliografia.

1. Geração de malha. 2. Tesselação de Delaunay. 3. Geometria computacional e topologia. 4. Algoritmos incrementais. 5. Sequências de inserção. I. Universidade Federal de Lavras. II. Título.

JÉSSICA RENATA NOGUEIRA

UMA AVALIAÇÃO DE SEQUÊNCIAS DE INSERÇÃO EM ALGORITMOS INCREMENTAIS PARA A TESSELAÇÃO DE DELAUNAY

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Inteligência Computacional e Processamento Digital de Imagens, para a obtenção do título de Mestre.

APROVADA em 17 de julho de 2015.

Professor Doutor Hermes Alves Filho	UERJ
Professor Doutor João Flávio Vieira de Vasconcellos	UERJ

Dr. Sanderson Lincohn Gonzaga de Oliveira

Orientador

LAVRAS - MG

2015

Dedico esta dissertação à minha amada irmã Bruna e aos meus amados pais: Sérgio e Rosangela.

Também dedico ao Pepe, meu amor de quatro patas. Vocês são o que tenho de melhor na vida. Amo vocês.

AGRADECIMENTOS

Agradeço a Deus, primeiramente, por todas as obras que vem realizando em minha vida. Agradeço a minha irmã e meu anjo em vida, Bruna, por ser sempre presente e fundamental em tudo que faço. Aos meus pais, por estarem sempre dispostos a me ajudar, por todo incentivo e pelo amor de sempre.

Agradeço ao meu orientador Sanderson, por toda ajuda e conselhos, que foram fundamentais para o desenvolvimento deste trabalho. Agradeço também aos amigos de laboratório e de mestrado, em especial ao Guilherme, pela amizade e pelas inúmeras ajudas. Agradeço a todos os funcionários do departamento de Ciência da Computação, especialmente à Nice, uma grande amiga durante toda a graduação e mestrado.

Agradeço ao meu namorado e amigo Breno, por estar sempre comigo e por me fazer acreditar que as coisas podem dar certo. Agradeço a minha irmã de consideração, Lívia, e ao Gustavo, pela amizade. Agradeço também ao meu amado avô Júlio.

Agradeço àqueles que estão no céu, olhando por mim e me abençoando. Em especial a minha amada avó Almerinda, minha eterna saudade e gratidão.

Agradeço também à CAPES pelo apoio financeiro.

RESUMO

Neste trabalho, são avaliadas 8 sequências de inserção de pontos em algoritmos incrementais para a geração da tesselação de Delaunay. Quatro dessas sequências são consideradas pela primeira vez: *H-Indexing*, espiral, rubro-negra em ordem e rubro-negra em largura. Essas sequências foram comparadas com: a sequência de inserção de pontos pela *cut-longest-edge kd-tree*; com a sequência dada pela curva de Hilbert; com a curva de Lebesgue; e também com sequência dada por inserção aleatória de pontos. Ao utilizar a biblioteca MPFR, foram testadas 6 distribuições de pontos no quadrado unitário e 7 distribuições de pontos no cubo unitário. Os algoritmos incrementais com as 4 sequências propostas neste trabalho não se mostraram competitivos com o algoritmo incremental com inserção de pontos dada pela *cut-longest-edge kd-tree*. Mais especificamente, o algoritmo incremental com inserção de pontos dada pela ordem da *cut-longest-edge kd-tree* apresentou os menores custos computacionais na geração das malhas, em todas as distribuições de pontos, em testes realizados em estruturas bidimensionais e tridimensionais.

Palavras-chave: geração de malha; tesselação de Delaunay; geometria computacional e topologia; algoritmos incrementais; sequências de inserção; distribuições não uniformes de pontos.

ABSTRACT

In this work, it is evaluated 8 insertion-point sequences in incremental algorithms to generate the Delaunay tessellation. Four of these sequences are considered for the first time: H-Indexing, spiral, red-black tree in-order and red-black-tree in level-order traversal. These sequences are compared with: point-insertion order given by cut-longest-edge kd-tree; with the order given by Hilbert space-filling curve; with Lebesgue space-filling curve and with the random point-insertion order. Using the GNU MPFR library, 6 dataset distributions were tested on unit square and 7 dataset distributions on the unit cube. The incremental algorithms with the 4 sequences that were proposed in this work are not competitive with the incremental algorithm using the point-insertion given by *cut-longest-edge kd-tree*. More specifically, the incremental algorithm using point-insertion sequence in the order given by the cut-longest-edge kd-tree, shows the lowest computational cost on mesh generation in tests carried out on 2D and on 3D.

Keywords: mesh generation; Delaunay tessellation; computational geometry and topology; incremental algorithms; insertion sequences; non-uniform point distributions.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Nesta seção, introduz-se o processo de geração de malhas e o contexto em que este trabalho está inserido nesse processo. Também, serão apresentados os objetivos e os passos realizados durante o desenvolvimento deste trabalho para que esses objetivos fossem alcançados.

Os tópicos abordados na introdução deste trabalho serão apresentados a seguir. Na subseção 1.1, serão apresentadas motivações e a contextualização em que este trabalho está inserido. Na subseção 1.2, será definido o problema de pesquisa abordado ao elaborar esta dissertação. Também, nessa subseção, serão apresentados os objetivos ao se realizar essa dissertação. Na subseção 1.3, será descrito como este trabalho está estruturado.

1.1 Contextualização e motivação

Esse trabalho está inserido no processo de geração de malhas, com enfoque na geração de malhas triangulares e tetraédricas. De maneira similar à apresentada por Floriani, Kobbelt e Puppo (2004), considere M como um conjunto finito de células conectadas no espaço Euclidiano E^d homeomórficas a um disco fechado. Então, M será uma *malha* d-dimensional se e, somente se, os interiores de qualquer par de células de dimensão d de M são disjuntos. Além disso, qualquer célula (d-dimensional) de M possui limite com, no mínimo, uma célula (d-dimensional) de M.

A geração de malhas compõe a vasta quantidade de temas abordados em geometria computacional e é uma área de pesquisa intensa, uma vez que

o uso de malhas pode contribuir em diversas áreas da ciência. Exemplos de aplicações que utilizam geração de malhas incluem processamento de imagem, sistemas de informação geográfica, análise de propagação de rádio, amostragem da população e interpolação multivariada.

Nas últimas décadas, o uso de malhas ocorreu, principalmente, devido à necessidade de modelar domínios computacionais para a simulação de problemas físicos e de engenharia, uma vez que existem situações em que encontrar uma solução exata para um problema específico não é possível ou é inviável. Métodos como o dos elementos finitos e dos volumes finitos são utilizados na simulação de fenômenos físicos, em que fluxo de fluidos, transferência de calor e propagação de onda eletromagnética são exemplos de fenômenos que podem ser simulados ao utilizar malhas.

A discretização de uma equação diferencial parcial em um domínio acarreta em um erro numérico. Em geral, esse erro depende do formato dos politopos que compõem a malha em que a equação diferencial parcial é resolvida. O termo politopo abrange uma ampla classe de objetos geométricos. Um politopo é um objeto geométrico com lados planos, que existe em quaisquer dimensões. Como exemplo, um polígono é um politopo em duas dimensões e um poliedro é um politopo em três dimensões.

A discretização de um domínio pode ocorrer utilizando malhas regulares ou malhas irregulares. Malhas regulares são compostas por politopos em que as relações de adjacência podem ser facilmente obtidas. Porém, esse tipo de discretização pode não ser eficiente para a representação de domínios descritos por geometrias complexas. Nesse caso, a utilização de uma malha irregular para a discretização do domínio pode ser a mais

adequada. Malhas irregulares são bastante versáteis por conta de sua capacidade em combinar diferentes formas de elementos com diversas formas de domínio. Em malhas irregulares, o tamanho dos elementos pode variar, de acordo com o domínio a ser discretizado. Porém, a utilização de malhas irregulares pode acarretar em grandes erros de aproximação se os elementos que compõem a malha forem de má qualidade.

O particionamento de uma descrição geométrica que representa um domínio físico em elementos que contêm o menor número de vértices é necessário em áreas diversas, em que modelagem dos sólidos, representação gráfica e computação gráfica são exemplos. Dessa maneira, malhas triangulares são as mais adequadas para a discretização de diversos domínios computacionais bidimensionais e malhas tetraédricas são úteis na discretização de domínios tridimensionais. Uma opção de malha irregular formada por triângulos, em estruturas bidimensionais, e tetraedros, em estruturas tridimensionais, e que possui aplicação em áreas diversas é a tesselação de Delaunay.

A tesselação de Delaunay tem sido utilizada no desenvolvimento de esquemas numéricos em uma grande variedade de disciplinas. Exemplos incluem aplicações médicas (PALIWAL et al., 2013) e video-games (GYVES; TOLEDO; RUDOMÍN, 2013). As versões bidimensionais e tridimensionais dessa malha foram estudadas neste trabalho e os custos computacionais e ocupações de memória obtidos ao se utilizar diferentes ordens de inserção de pontos em algoritmos incrementais são apresentados.

Uma tesselação de Delaunay é uma estrutura que possui propriedades matemáticas úteis para a criação de malhas triangulares e

tetraédricas de qualidade. Em duas dimensões, para um conjunto fixo de pontos, essa triangulação tem a propriedade de maximização do ângulo mínimo. Essa propriedade otimiza critérios geométricos relacionados à precisão da interpolação e torna a triangulação de Delaunay aplicável em áreas diversas.

1.2 Problema e objetivos da pesquisa

Existem diversos algoritmos propostos para a geração da tesselação de Delaunay. Para uma revisão de algoritmos lineares para a geração das tesselações de Delaunay, em duas e em três dimensões, e do diagrama de Voronoi, veja Oliveira, Nogueira e Tavares (2014).

Dentre os algoritmos para a geração dessas malhas, Liu, Yan e Lo (2013) apresentaram um algoritmo que está, possivelmente, no estado da arte para a geração dessa malha em estruturas tridimensionais. Já o algoritmo proposto por Lo (2013) é o provável estado da arte na geração da triangulação de Delaunay. Porém, há a possibilidade de que sejam propostos novos algoritmos para a geração dessa tesselação e que esses algoritmos sejam mais eficientes que os algoritmos que estão no estado da arte atualmente.

Os algoritmos propostos para a construção da tesselação de Delaunay utilizam abordagens diversas. Exemplos de abordagens são de algoritmos que utilizam divisão e conquista, *lift mapping* e inserção incremental de pontos. Em algoritmos incrementais, são diversas as maneiras propostas que tratam como ocorrerá a inserção de pontos no domínio. Isso porque o tempo para a construção da tesselação de Delaunay

em um algoritmo incremental está diretamente relacionado à ordem em que os pontos são inseridos.

Diferentes formas de inserção de pontos no domínio para a geração da tesselação de Delaunay serão avaliadas neste trabalho. Será investigado se a utilização das curvas de preenchimento de Lebesgue e Hilbert, do esquema de indexação *H-indexing*, da ordem espiral, da inserção aleatória de pontos e da ordem da árvore rubro-negra, utilizando os percursos em ordem e em largura, para determinar a ordem em que os pontos serão inseridos, poderão apresentar vantagens em relação aos algoritmos que estão no estado da arte para a tesselação de Delaunay. Exemplos para essas vantagens são a obtenção de um custo computacional menor ou uma utilização de memória menor pelos novos algoritmos.

1.3 Estrutura do trabalho

Este trabalho está dividido em quatro partes, que são: referencial teórico, metodologia e infraestrutura, resultados, para estruturas bidimensionais e tridimensionais e conclusão. As subseções estão organizadas da maneira apresentada a seguir.

Na seção 2, é apresentado o referencial teórico necessário para a compreensão deste trabalho. Na seção 3, são apresentadas a metodologia e a infraestrutura necessárias para a realização deste trabalho. Nas seções 4 e 5, são apresentados, respectivamente, os resultados obtidos ao se utilizar as diferentes ordens de inserção de pontos em algoritmos incrementais em estruturas bidimensionais e tridimensionais. Finalmente, na seção 6, conclusões obtidas após a realização desse trabalho serão apresentadas.

2 REFERENCIAL TEÓRICO

Nesta seção, serão apresentados os conceitos básicos relacionados à tesselação de Delaunay e os conceitos relacionados à sua estrutura dual, que é o diagrama de Voronoi. Também, será apresentado o algoritmo de Bowyer-Watson e as ordens de inserção de pontos nos algoritmos incrementais que serão implementados neste trabalho.

Na subseção 2.1, apresenta-se a tesselação de Delaunay. Na subseção 2.2, é apresentado o diagrama de Voronoi, que é o dual geométrico da tesselação de Delaunay. Na subseção 2.3, é apresentado o algoritmo de Bowyer-Watson, que é um algoritmo incremental utilizado para a geração da tesselação de Delaunay.

Nas demais subseções, são apresentadas as ordens de inserção de pontos em algoritmos incrementais utilizadas neste trabalho. Na subseção 2.4, é apresentado o esquema de inserção de pontos proposto por Liu, Yan e Lo (2013), para a geração da tesselação de Delaunay. Na subseção 2.5, são conceitualizadas curvas de preenchimento de espaço. As curvas de preenchimento de espaço de Hilbert e de Lebesgue, que foram utilizadas neste trabalho, são apresentadas nessa mesma subseção. Nas subseções 2.6, 2.7 e 2.8, são apresentados, respectivamente, o indexador *H-indexing*, a ordem espiral e a árvore rubro-negra, que são ordens de inserção de pontos em algoritmos incrementais que também foram utilizadas neste trabalho.

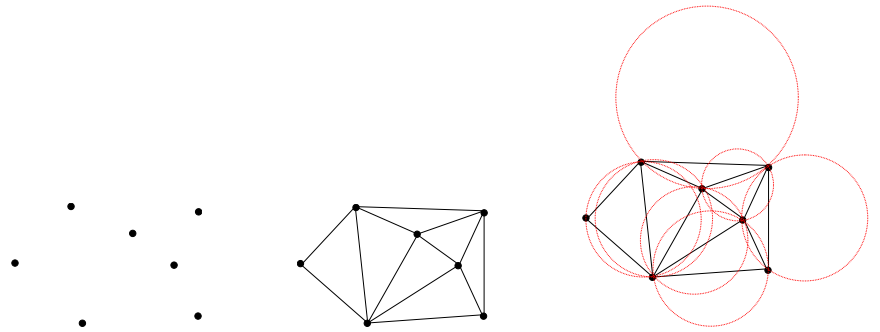
2.1 Tesselação de Delaunay

Algumas das descrições apresentadas nesta seção também podem ser encontradas em Nogueira e Oliveira (2011a, 2011b, 2012). Esse trabalho se restringe a malhas bidimensionais e tridimensionais, embora as definições apresentadas a seguir possam ser estendidas para estruturas de dimensões aleatórias.

Uma tesselação de Delaunay (TD), para um conjunto P de pontos, deve atender à condição de Delaunay, que é: $TD(P)$ é uma tesselação de Delaunay se nenhum circuncírculo (em estruturas bidimensionais) ou circunsfera (em estruturas tridimensionais) que passa pelos elementos de TD(P) contém pontos em seu interior.

Na Figura 1, observa-se uma triangulação de Delaunay para um conjunto de 7 pontos. Na Figura 1(a), apresenta-se o conjunto inicial de pontos que será utilizado para a construção da triangulação de Delaunay. Na Figura 1(b), pode-se observar a triangulação de Delaunay gerada para esse conjunto de vértices. Na Figura 1(c), pode-se observar cada um dos triângulos da malha de Delaunay com seus circuncírculos correspondentes. Um exemplo para uma tetraedrização de Delaunay gerada em um conjunto de 50 pontos pode ser observada na Figura 2.

Na triangulação de Delaunay, ocorre a maximização do ângulo mínimo de todo triângulo da triangulação e essa propriedade é conhecida como MaxMin. A propriedade MaxMin foi inicialmente notada por Lawson (1977) e contribuiu na popularização da triangulação de Delaunay. A propriedade MaxMin, segundo Shewchuk (1997b), não pode ser estendida



(a) Conjunto inicial de pontos.

(b) Triangulação de Delaunay.

(c) Triângulos de Delaunay e circuncírculos correspondentes.

Figura 1 Triangulação de Delaunay para um conjunto de sete pontos.

para dimensões maiores que dois (NOGUEIRA, 2013).

Essa triangulação (ou tetraedrização, em três dimensões) possui a propriedade de unicidade. Na propriedade de unicidade, dado conjunto de vértices, tem-se que é possível gerar apenas uma triangulação (tetraedrização, em três dimensões) de Delaunay para esse conjunto. Essa propriedade não é válida quando quatro (cinco, em três dimensões) ou mais pontos são cocirculares (coesféricos, em três dimensões). Para um caso como esse, as triangulações (tetraedrizações, em três dimensões) que podem ser geradas satisfazem a condição de Delaunay.

De maneira similar a Nogueira (2013), uma triangulação (tetraedrização) é estritamente de Delaunay se todas as arestas (faces) que a compõe são *arestas locais* (*faces locais*) de Delaunay. Uma aresta (face) de Delaunay é local se:

- pertencer somente a um triângulo (tetraedro) e este triângulo

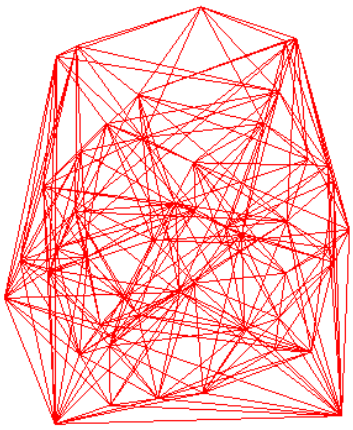


Figura 2 Tetraedrização de Delaunay para 50 pontos.

(tetraedro) pertencer à fronteira do fecho convexo da triangulação (tetraedrização); ou

- é compartilhada por dois triângulos (três tetraedros), digamos $\triangle abc$ e $\triangle abe$ ($\triangle abce$ e $\triangle abcf$) e o ponto e (ou f, na estrutura tridimensional) está fora do circuncírculo (da circunsfera) que passa por a, b e c (a, b, c e e).

Para determinar se um ponto é interior ao circuncírculo, em estruturas bidimensionais, ou à circunsfera, em estruturas tridimensionais, os testes do circuncírculo, da circunsfera e testes de orientação devem ser utilizados. Esses testes são apresentados a seguir e foram retirados de Shewchuk (1997a). A seguir, será apresentada uma breve descrição desses testes.

Para verificar se um ponto é interior a um triângulo, é necessário

que os pontos do triângulo t estejam em sentido anti-horário. O teste em que se verifica se os pontos a, b e c de t estão em sentido anti-horário, em duas dimensões, é apresentado no *teste de orientação bidimensional*(a,b,c), que é dado a seguir.

$$\textit{Teste de orientação bidimensional}(a,b,c) = \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix}$$

Caso o resultado do *teste de orientação bidimensional(a,b,c)* seja positivo, os pontos de t estão em sentido anti-horário. Se for negativo, os pontos de t estão em sentido horário. Senão, o resultado é igual a zero e os pontos são colineares. Considerando que os pontos a, b e c de um triângulo t estejam em sentido anti-horário, o teste do circuncírculo utilizado para verificar se um ponto e é interior ao circuncírculo que passa por t é apresentado a seguir. O teste do circuncírculo foi proposto por Guibas e Stolfi (1985).

$$\textit{Teste do circuncírculo}(a,b,c,e) = \begin{vmatrix} x_a & y_a & x_a^2+y_a^2 & 1 \\ x_b & y_b & x_b^2+y_b^2 & 1 \\ x_c & y_c & x_c^2+y_c^2 & 1 \\ x_e & y_e & x_e^2+y_e^2 & 1 \end{vmatrix}$$

No *teste do circuncírculo(a,b,c,e)*, um valor positivo é retornado se e é interior ao circuncírculo que passa por t, é negativo se e é exterior ao

circuncírculo que passa por t e é igual a zero se os pontos a, b, c e e são cocirculares.

Os testes de orientação e do circuncírculo podem ser estendidos para estruturas tridimensionais. Em três dimensões, o teste de orientação dos pontos é dado da maneira apresentada a seguir.

$$\textit{Teste de orientação tridimensional(a,b,c,e)} = \begin{vmatrix} x_a & y_a & z_a & 1 \\ x_b & y_b & z_b & 1 \\ x_c & y_c & z_c & 1 \\ x_e & y_e & z_e & 1 \end{vmatrix}$$

O resultado para o *teste de orientação tridimensional(a,b,c,e)* é maior que zero caso os pontos a, b, c e e de um tetraedro *tet* estejam em sentido anti-horário. O resultado para o *teste de orientação tridimensional(a,b,c,e)* é menor que zero caso os pontos a, b, c e e de um tetraedro *tet* estejam em sentido horário. Senão, o resultado do *teste de orientação tridimensional(a,b,c,e)* é igual a zero e os pontos de *tet* são coplanares.

Considerando que os pontos estão em sentido anti-horário, o teste da circunsfera é realizado para verificar se um ponto e é interior à circunsfera que passa por *tet*. O teste da circunsfera é apresentado a seguir.

$$\textit{Teste da circunsfera(a,b,c,e,f)} = \begin{vmatrix} x_a & y_a & z_a & x_a^2+y_a^2+z_a^2 \\ x_b & y_b & z_b & x_b^2+y_b^2+z_b^2 \\ x_c & y_c & z_c & x_c^2+y_c^2+z_c^2 \\ x_e & y_e & z_e & x_e^2+y_e^2+z_e^2 \\ x_f & y_f & z_f & x_f^2+y_f^2+z_f^2 \end{vmatrix}$$

No *teste da circunsfera(a,b,c,e,f)*, um valor positivo é retornado se f é interior ao circuncírculo que passa por *tet*, é negativo se f é exterior ao circuncírculo que passa por *tet* e é igual a zero se os pontos a, b, c, e e f são cocirculares.

Algoritmos que utilizam a triangulação de Delaunay em simulações podem ser encontrados, por exemplo, em Antony e Vigila (2013) e Thirusittampalam et al. (2013). Essas aplicações da triangulação de Delaunay foram, respectivamente, em segmentação de imagens de células e para um sistema de criptografia biométrica. Simulações como essas podem ser beneficiadas quando se utilizam algoritmos para a geração da tesselação de Delaunay com custo computacional baixo ou com baixa ocupação de memória.

Ainda, a triangulação de Delaunay pode ser gerada por *softwares* comerciais como o Matlab[1] e ANSYS ICEM CFD[2]. Outros softwares que

[1]Matlab. Disponível em: http://www.mathworks.com/products/matlab/. Data de acesso: 13 ago. 2015.

[2]ANSYS ICEM CFD. Disponível em: http://www.ansys.com. Data de acesso: 13 ago. 2015.

geram a triangulação de Delaunay são o DelPSC[3], GTS[4] e o QualMesh[5]. A biblioteca *The Computational Geometry Algorithms Library* (CGAL) [6] é uma das bibliotecas mais utilizadas para a geração da tesselação de Delaunay e do diagrama de Voronoi. Como exemplo, Liu, Yan e Lo (2013) utilizam essa biblioteca para a geração da tetraedrização de Delaunay. Por padrão, a CGAL utiliza a ordem da curva de Hilbert para definir a ordem em que os pontos serão inseridos no domínio.

Existem diversos algoritmos para a geração da tesselação de Delaunay. Esses algoritmos começaram a ser propostos na década de 1970. De acordo com Cheng, Dey e Shewchuk (2013), o processo de geração de malhas não estruturadas para o método dos elementos finitos começou em 1970 com o artigo de Frederick, Wong e Edge (1970). De acordo com Cheng, Dey e Shewchuk (2013), no artigo de Frederick, Wong e Edge (1970), descreve-se o primeiro algoritmo para a geração da malha de Delaunay, o primeiro método de avanço de fronteira (*advancing front method*, que é um método em que a estrutura final é obtida pela adição progressiva de elementos nas bordas da malha), e o primeiro algoritmo para a malha de Delaunay no plano. De acordo com Cheng, Dey e Shewchuk (2013), o fato irônico é que os autores desse artigo parecem não ter conhecimento de que a triangulação que eles criaram era a triangulação de Delaunay restrita, uma variação da triangulação de Delaunay. De acordo com Cheng, Dey e Shewchuk (2013), talvez esse artigo não seja bastante conhecido por estar

[3]DelPSC. Disponível em: http://www.cse.ohio-state.edu/ tamaldey/delpsc.html. Data de acesso: 13 ago. 2015.

[4]GTS. Disponível em: http://gts.sourceforge.net/index.html. Data de acesso: 13 ago. 2015.

[5]QualMesh. Disponível em: http://www.cse.ohio-state.edu/ tamaldey/qualmesh.html. Data de acesso: 13 ago. 2015.

[6]CGAL. Disponível em: https://www.cgal.org/. Data de acesso: 13 ago. 2015.

duas décadas à frente de seu tempo. Para uma apresentação de algoritmos que geram a triangulação de Delaunay e de algoritmos que geram o diagrama de Voronoi, que é a estrutura dual da triangulação de Delaunay e que será descrito na seção a seguir, veja Nogueira (2013).

A construção da triangulação de Delaunay ocorre, no pior caso, em tempo O(n lg n). Porém, em 2003, com a proposta do algoritmo BRIO (Biased Randomized Insertion Order (AMENTA; CHOI; ROTE, 2003)), começou-se a levar em consideração a ordem em que os pontos são inseridos no domínio, a fim de ocorrer um maior número de acertos em *cache*. Na Figura 3, observa-se um exemplo de como os pontos são inseridos no BRIO.

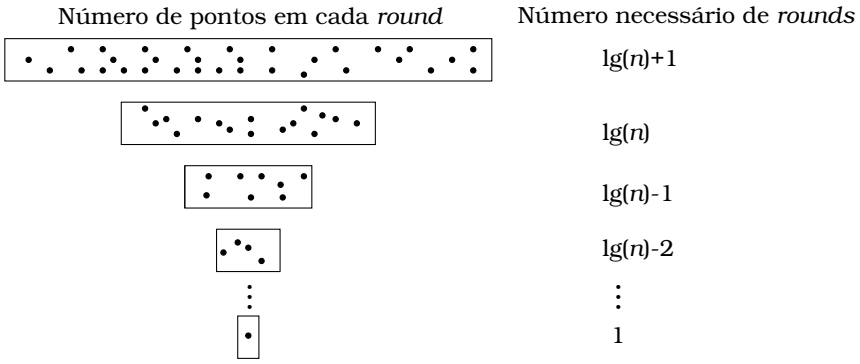


Figura 3 Exemplificação de como ocorrem as inserções de pontos ao utilizar o algoritmo BRIO.

Baseados na ideia de um número maior de acertos em memória cache e na não aleatoriedade dos pontos de entrada, Liu e Snoeyink (2005) apresentaram, em 2005, um algoritmo incremental para a geração da tetraedrização de Delaunay em que os pontos são inseridos na ordem da curva de Hilbert. Apesar de diversas propostas e simulações nos anos seguintes, provavelmente até o ano de 2013, esse algoritmo era o estado da arte para a geração de tesselações de Delaunay. Em 2013, Liu, Yan e Lo

(2013) apresentaram um algoritmo incremental com desempenho superior ao apresentado pelo algoritmo incremental utilizando a curva de Hilbert para a geração da tesselação de Delaunay. É possível que o algoritmo de Liu, Yan e Lo (2013) esteja no estado da arte para a geração da tetraedrização de Delaunay.

Lo (2013) melhorou o esquema de Liu, Yan e Lo (2013) e propôs um esquema de inserção *multi-grid*. De acordo com Lo (2013), esse esquema resultou em um algoritmo com comportamento aproximadamente linear, para conjuntos de dados em que os pontos estão distribuídos de maneira uniforme. Ainda, de acordo com Lo (2013), esse esquema de inserção *multi-grid* foi mais estável e eficiente quando comparado aos esquemas de inserção utilizando a ordem da *cut-longest-edge kd-tree* como também dos resultados obtidos ao utilizar um esquema de subdivisão regular do domínio. De acordo com o mesmo autor, o esquema de divisão do domínio por malhas regulares foi bastante sensível à distribuição de pontos e o esquema de inserção de pontos em algoritmos incrementais utilizando a ordem da *cut-longest-edge kd-tree* teve custo computacional alto para malhas com triângulos alongados. Lo (2013) realizou teste com até 100 milhões de pontos em diferentes distribuições de pontos. Para maiores detalhes, veja Oliveira, Nogueira e Tavares (2014).

2.2 Diagrama de Voronoi

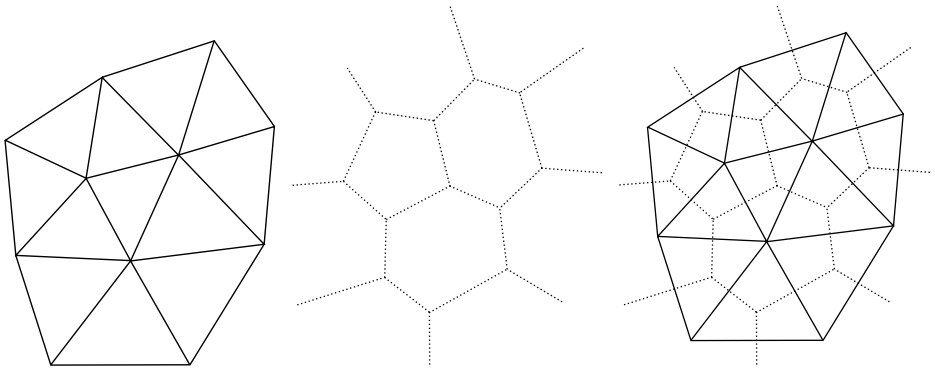
O diagrama de Voronoi foi nomeado dessa maneira por Delaunay (1934). Delaunay (1934) dedicou suas notas sobre *esferas vazias* a Voronoi (1908); porém, o nome tesselação de Dirichlet também pode vir a ser

empregado para fazer referência a essa estrutura e os polígonos de Vononoi também podem ser nomeados como regiões de Dirichlet ou como politopos de Thiessen. Nesta dissertação, será utilizada apenas a denominação de diagrama de Voronoi e de polígonos de Voronoi. Um diagrama de Voronoi é um tipo específico de decomposição de um espaço e é utilizado em áreas diversas, tais como epidemiologia, geofísica, meteorologia, computação gráfica e robótica (NOGUEIRA, 2013).

O diagrama de Voronoi é determinado pelas distâncias para um conjunto de pontos no espaço. Considere $S \subseteq \mathbb{R}^d$ como um conjunto de n pontos geradores p_i no espaço Euclidiano, para $1 \leq i \leq n$. O politopo de Voronoi P_i de um ponto gerador p_i é o conjunto de todos os pontos no $\mathbb{R}^d$ que são mais próximos a p_i do que a qualquer ponto gerador em S. Formalmente, para uma dimensão d maior ou igual a 2, cada politopo é um conjunto de pontos $P_i = \{p \in \mathbb{R}^d : (\exists p_i \in S)(\forall p_j \in S)\ ||p - p_i|| \leq ||p - p_j||,$ com $i \neq j \wedge 1 \leq i, j \leq n\}$ (OLIVEIRA; NOGUEIRA; TAVARES, 2014).

O diagrama de Voronoi é definido como uma estrutura dual da tesselação de Delaunay. Dessa maneira, é possível a transformação da triangulação (tetraedrização) de Delaunay em diagrama de Voronoi e vice-versa. Na Figura 4, pode ser observada uma partição do diagrama de Voronoi e a triangulação de Delaunay correspondente.

O diagrama de Voronoi é utilizado em simulações diversas. Wood (2013), por exemplo, utiliza o diagrama de Voronoi na modelagem de células. Também, há patentes de algoritmos que utilizam o diagrama de Voronoi para suas aplicações. Continuous... (2013), por exemplo, utiliza o diagrama de Voronoi no esboço de nó de distribuição física.



(a) Triangulação de Delaunay. (b) Diagrama de Voronoi. (c) Triangulação de Delaunay e diagrama de Voronoi correspondente.

Figura 4 Representação da dualidade entre a triangulação de Delaunay e do diagrama de Voronoi. Adaptado de Nogueira (2013).

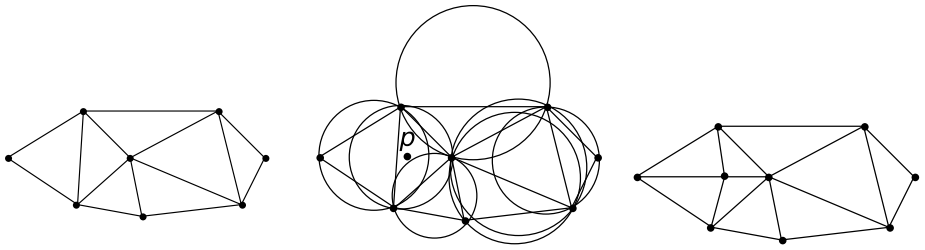
2.3 Algoritmo de Bowyer-Watson

Esse algoritmo é chamado de algoritmo de Bowyer-Watson pois Bowyer (1981) e Watson (1981) publicaram, simultaneamente, em uma mesma edição, de uma mesma revista, algoritmos similares para a geração das malhas de Voronoi e de Delaunay. O algoritmo de Bowyer (1981) foi proposto para a geração do diagrama de Voronoi e o algoritmo de Watson (1981) foi proposto para a tesselação de Delaunay. Neste trabalho, o algoritmo de Bowyer-Watson é um utilizado na geração da tesselação de Delaunay.

O algoritmo de Bowyer-Watson é incremental, ou seja, a cada iteração, um ponto p é inserido na malha de Delaunay. Nesse algoritmo, após a inserção de p, são verificados circuncírculos (ou circunsferas) que contêm p. Para verificar se um circuncírculo (em estruturas bidimensionais) contém p, os testes de orientação e do circuncírculo devem ser utilizados.

Em estruturas tridimensionais, para realizar essa verificação, os testes de orientação e da circunsfera devem ser utilizados. Esses testes são apresentados na subseção 2.1.

Caso um circuncírculo que passa por um triângulo (ou uma circunsfera que passa por um tetraedro, em estruturas tridimensionais) contenha p, esse triângulo (tetraedro) deve ser armazenado. Os triângulos (tetraedros) armazenados são removidos da triangulação (tetraedrização) e novos triângulos (tetraedros) são formados pela conexão de p com elementos da borda da cavidade formada após a remoção dos triângulos (tetraedros). Os passos que devem ser executados após a inserção de um ponto na triangulação de Delaunay podem ser observados na Figura 5.



(a) Triangulação inicial de Delaunay.

(b) Inserção do ponto p na triangulação de Delaunay.

(c) Triângulos com o circuncírculo invadido são removidos e triângulos válidos para a malha de Delaunay são criados.

Figura 5 Exemplificação do algoritmo de Bowyer-Watson

Um pseudocódigo apresentando os passos necessários para a geração da tesselação de Delaunay utilizando o algoritmo de Bowyer-Watson é apresentado na Figura 18. Esse algoritmo recebe um conjunto de pontos P e esse conjunto será utilizado para a construção da tesselação de Delaunay. O rearranjo dos pontos de entrada, apresentado na linha 2 desse pseudocódigo,

ocorre, neste trabalho, utilizando a ordem das curvas de preenchimento de espaço de Lebesgue e de Hilbert, da ordem da árvore rubro-negra, com os percursos em ordem ou em largura, da ordenação espiral, do esquema de indexação *H-indexing*, do percurso em largura na *cut-longest-edge kd-tree* ou através da inserção aleatória dos pontos na malha. Esses esquemas de inserção de pontos são apresentados nas subseções a seguir.

Entrada: conjunto de pontos $P = \{p_1, p_2, \cdots, p_n\}$.
Saída: tesselação de Delaunay TD(P).

```
1  início
2      rearranje os pontos de entrada p_1, p_2, ..., p_n;
3      construa st, que é um supertriângulo (tetraedro) que
       contém todos os os pontos de P;
4      insira st em TD(P); i ← 1;
5      enquanto (i ≤ |P|) faça
6          list_t ← ∅;
7          localize um triângulo (tetraedro) t cujo circuncírculo
           (circunsfera) contém p_i;
8          insira t em list_t;
9          para cada (triângulo (tetraedro) t_a adjacente aos
           triângulos (tetraedros) armazenados em list_t) faça
10             se (o circuncírculo (circunsfera) de t_a contém p_i)
               então insira t_a em list_t;
11         fim para cada
12         remova de TD(P) triângulos (tetraedros) armazenados
           em list_t;
13         insira, em TD(P), triângulos (tetraedros) formados pela
           conexão de p_i com elementos da borda da cavidade
           formada após a remoção de triângulos (tetraedros)
           armazenados em list_t;
14         i ← i+1;
15     fim enquanto
16     remova de TD(P) os triângulos (tetraedros) que contêm
       vértices do supertriângulo (supertetraedro);
17     retorna TD(P);
18 fim
```

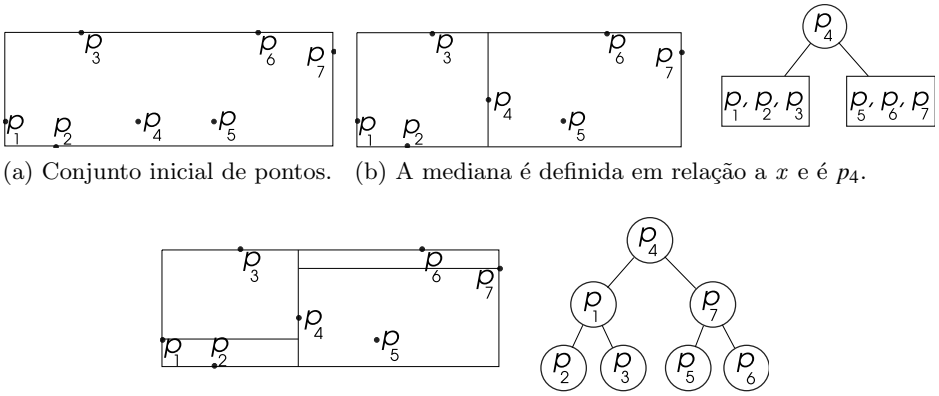
Figura 6 Algoritmo incremental para gerar a malha de Delaunay.

2.4 Algoritmo de Liu, Yan e Lo

Uma estrutura de dados utilizada para o particionamento do espaço em dimensões aleatórias é a árvore de dimensão k, denominada *kd-tree*, que foi proposta por Bentley (1975). De maneira genérica, uma *kd-tree* é uma árvore binária cujos nós que possui têm k coordenadas. Exceto pelos nós folhas, cada nó dessa árvore gera, implicitamente, um hiperplano que divide o espaço em duas partes, que são conhecidos como semiespaços.

Para essa árvore bidimensional (tridimensional), cada nó tem as coordenadas x e y (x, y e z) e, para a construção dessa árvore, inicialmente, subdivide-se o plano na mediana do eixo x; em seguida, subdividem-se os semiplanos em relação à mediana do eixo y. Em estruturas tridimensionais, subdivide-se o espaço na mediana do eixo x; em seguida, subdividem-se os semiespaços em relação à mediana do eixo y e, finalmente, subdividem-se os semiespaços em relação à mediana do eixo z. Esse processo continua de forma que sejam realizadas rotações que alternam o eixo x, o eixo y e o eixo z (em estruturas tridimensionais) para a realização das subdivisões. O processo de divisão dos semiespaços ocorre até que haja apenas um ponto no semiespaço. Não é possível construir uma *kd-tree* cujos dados de entrada possuam diferentes dimensões; ou seja, todos os dados armazenados na *kd-tree* devem ter o mesmo número de coordenadas. Na Figura 7, pode-se observar um exemplo, para um conjunto de sete pontos e em dimensão dois, para a construção da *kd-tree*.

Na Figura 8, é apresentado um pseudocódigo de como é realizada a construção da *kd-tree* em estruturas bidimensionais. Esse pseudocódigo pode ser estendido para uma *kd-tree* de dimensão aleatória. Note que, a



(a) Conjunto inicial de pontos. (b) A mediana é definida em relação a x e é p_4.

(c) A mediana dos subplanos, à esquerda, é p_1, e, à direita, é p_7 e é definida em relação ao eixo y. Os nós p_2, p_3, p_5 e p_6 são folhas da *kd-tree*.

Figura 7 Construção da *kd-tree*.

cada chamada recursiva desse algoritmo, o valor da variável nC deve ser incrementado e esse valor deve ser inicializado em 0.

Liu, Yan e Lo (2013) modificaram a estrutura *kd-tree* e obtiveram bons resultados no custo computacional de algoritmos incrementais para a geração da tetraedrização de Delaunay. A modificação para essa estrutura é denominada *cut-longest-edge kd-tree*. Nesse algoritmo, os pontos são inseridos no domínio de acordo com o percurso em largura na *cut-longest-edge kd-tree*.

De acordo com os resultados apresentados por Liu, Yan e Lo (2013), o desempenho do algoritmo utilizando a ordem de inserção de pontos dada pela *cut-longest-edge kd-tree* apresentou desempenho superior ao desempenho apresentado pelos algoritmos que utilizaram a ordem de inserção de pontos na ordem da curva de Hilbert e dos algoritmos utilizando o BRIO, especialmente para distribuições não uniformes de pontos. Além

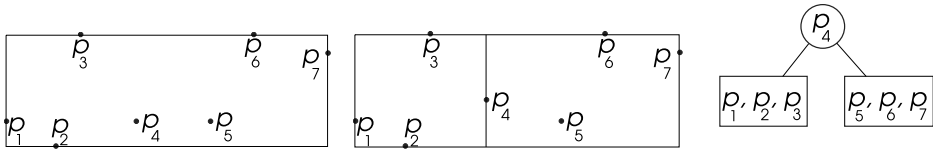
```
   Entrada: conjunto de pontos P; nível corrente, nC, da kd-tree.
   Saída: nó corrente v da kd-tree.
 1 início
 2     se (|P| = 1) então retorna v;
 3     senão
 4         se (nC % 2 = 0) então ordene P pelas coordenadas x
           dos pontos;
           // se o resto da divisão de nC é igual a zero,
              ordene P pelas coordenadas dos pontos em x
 5         senão ordene P pelas coordenadas y dos pontos;
           // caso contrário, se o resto da divisão de nC
              é igual a um, ordene P pelas coordenadas
              dos pontos em y
 6         v ← ponto na mediana de P;
 7         P1 ← conjunto de pontos que estão à esquerda de v;
 8         P2 ← conjunto de pontos que estão à direita de v;
 9         v_left ← ConstróiKdTree(P1, nC+1);
10         v_right ← ConstróiKdTree(P2, nC+1);
11     fim se
12     retorna v;
13 fim
```

Figura 8 ConstróiKdTree.

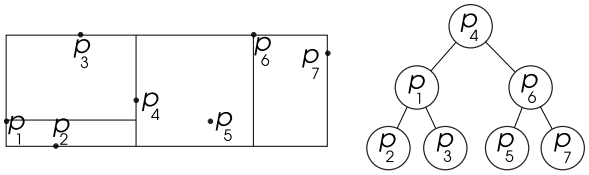
disso, de acordo com Liu, Yan e Lo (2013), o custo computacional apresentado ao utilizar a ordem da *kd-tree* original para determinar a ordem em que os pontos são inseridos no domínio é superior ao custo computacional ao se utilizar a inserção de pontos na ordem da *cut-longest-edge kd-tree*.

Na *cut-longest-edge kd-tree*, constrói-se uma caixa delimitadora (*bounding box*) e realiza-se o mesmo esquema de corte que é realizado na *kd-tree* original. Porém, o esquema de corte ocorre na maior aresta do domínio. De acordo com Liu, Yan e Lo (2013), esse esquema permite que sejam criadas regiões de tamanho mais homogêneo que o esquema da *kd-tree*

original, pois se consideram as diferentes dimensões da estrutura. Na Figura 9, pode-se observar um exemplo para a construção da *cut-longest-edge kd-tree* em dimensão dois.



(a) Conjunto inicial de pontos. (b) A maior diferença entre as coordenadas ocorre em x, cuja mediana é p_4.



(c) A maior diferença à esquerda é em y e, à direita, é em x, cujas medianas são p_1 e p_6. Os demais nós são folha.

Figura 9 Construção da *cut-longest-edge kd-tree*.

2.5 Curvas de preenchimento de espaço

Uma *curva de preenchimento de espaço* é uma curva em que todo o seu intervalo percorre um hipercubo d-dimensional completo. Cantor (1878) demonstrou que havia uma correspondência de um-para-um entre o intervalo unitário e o quadrado unitário e questionou sobre a possibilidade de um intervalo [0,1] poder ser mapeado, bijetivamente, no quadrado unitário. Eugen Netto (1879) mostrou que tal mapeamento bijetivo é, necessariamente, descontínuo. Desconsiderando-se a questão da bijeção, a questão tornou-se sobre a possibilidade de se obter um mapeamento sobrejetor contínuo de [0,1] em um quadrado unitário (SAGAN, 1994). Um

mapeamento será sobrejetor caso o contradomínio do mapeamento seja igual ao seu conjunto imagem.

Peano (1890) descobriu uma curva de preenchimento de espaço que gera um caminho hamiltoniano nas partições do quadrado unitário. A curva de Peano representou a resposta sobre a possibilidade de haver um mapeamento sobrejetor contínuo em um intervalo do domínio.

Hilbert (1891) descobriu outra curva de preenchimento de espaço. A curva de Peano foi definida de maneira puramente analítica e a abordagem de Hilbert foi de maneira geométrica. Muitas outras curvas de preenchimento foram descobertas após a curva de Hilbert. Exemplos para essas curvas são as de Lebesgue (1904), Sierpiński (1912) e de Gosper. A curva de Gosper é uma curva de preenchimento de espaço descoberta por William Gosper, em 1973, e que foi introduzida por Gardner (1976). Informações detalhadas sobre curvas de preenchimento de espaço podem ser encontradas em Sagan (1994). A seguir, as curvas de preenchimento de espaço de Hilbert e de Lebesgue são apresentadas.

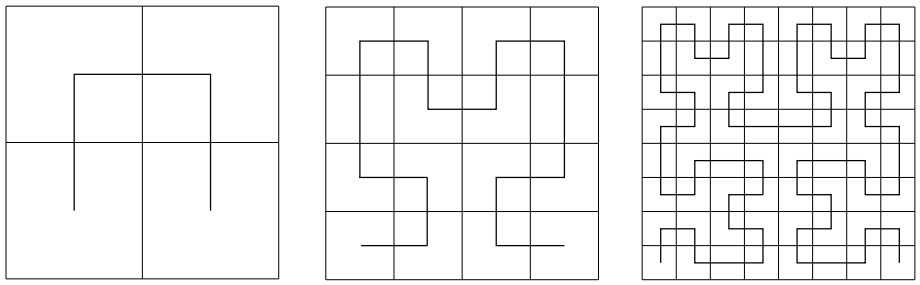
2.5.1 Curva de Hilbert

A curva de Hilbert (1891) é uma curva de preenchimento de espaço d-dimensional. De acordo com Haverkort (2011), embora as propriedades da curva de Hilbert permitam que a mesma seja única em duas dimensões, não está claro qual é a melhor maneira de generalizar essa curva para espaços de dimensão d.

Essa curva é construída de maneira recursiva e, a cada iteração, em casos bidimensionais, a curva de Hilbert contém quatro cópias da curva da

iteração anterior e as quatro cópias da curva são rotacionadas de forma que cada saída de uma curva possa ser unida à entrada da próxima. Essa curva bidimensional permite que os objetos a serem ordenados possuam uma boa localidade entre si, o que torna a curva de Hilbert útil em diversas aplicações.

Ao propor essa curva, em estruturas bidimensionais, Hilbert (1891) provou que um intervalo I pode ser mapeado de maneira contínua em um quadrado Q. Com isso, após particionar Q em quatro subquadrados congruentes, cada subintervalo pode ser mapeado, de maneira contínua, em um dos subquadrados. Caso essas subdivisões ocorram no infinito, I e Q serão particionados em 2^{2n} réplicas congruentes, para $n = 1, 2, \cdots$. Dessa maneira, Hilbert (1891) demonstrou que os subquadrados podem ser rearranjados de forma que subintervalos adjacentes correspondam a subquadrados adjacentes com uma aresta em comum e as relações entre os subquadrados sejam mantidas Sagan (1994). Na Figura 10, pode-se observar três níveis da curva de Hilbert bidimensional.



(a) Curva de Hilbert com um nível de refinamento.

(b) Curva de Hilbert com dois níveis de refinamento.

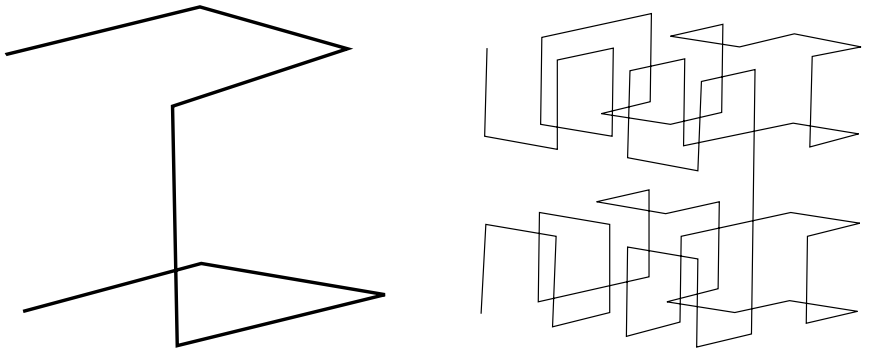
(c) Curva de Hilbert com três níveis de refinamento.

Figura 10 Curva de Hilbert bidimensional.

De acordo com Haverkort (2011), as propriedades que fazem com

que a curva de Hilbert seja única em duas dimensões, não são válidas para estruturas tridimensionais. Segundo o mesmo autor, as propriedades da curva de Hilbert tornam possíveis a construção de 10.694.807 curvas de preenchimento de espaço estruturalmente diferentes em três dimensões.

Na curva de Hilbert tridimensional, o cubo original é subdividido em oito subcubos e a curva de Hilbert deve ser rotacionada de forma que a saída de um subcubo represente a entrada do próximo subcubo. Caso essas subdivisões ocorram no infinito, o intervalo será particionado em 2^{3n} réplicas congruentes, para $n = 1, 2, \cdots$. Na Figura 11, pode-se observar um exemplo da curva de Hilbert tridimensional.



(a) Curva de Hilbert com um nível de refinamento.

(b) Curva de Hilbert com dois níveis de refinamento.

Figura 11 Curva de Hilbert tridimensional.

A curva de Hilbert foi empregada, a partir da versão 4, na Computational Geometry Algorithms Library - CGAL (2012), para a ordenação espacial de pontos inseridos na tesselação de Delaunay. Na CGAL (2012), utiliza-se a curva de Hilbert e o algoritmo BRIO.

2.5.2 Algoritmo de Liu e Snoeyink

O algoritmo incremental de Liu e Snoeyink (2005) foi proposto para a geração da tetraedrização de Delaunay em conjuntos de pontos bem distribuídos no domínio. Até 2013, quando foi proposto o algoritmo incremental utilizando a ordem da *cut-longest-edge kd-tree* para a inserção dos pontos, o algoritmo de Liu e Snoeyink (2005) era o provável estado da arte para a geração de tesselações de Delaunay, tanto em estruturas bidimensionais quanto em estruturas tridimensionais.

Nesse algoritmo, a ordem da curva de Hilbert determina a sequência em que os pontos serão inseridos no domínio. Para ordenar um conjunto de vértices pela ordem dessa curva, nesse algoritmo, subdivide-se o domínio em 2^{3i} subdivisões, em que i é grande o suficiente para que poucos passos recursivos, ou seja, novas subdivisões mais internas das subdivisões, sejam necessários. Dessa maneira, vértices próximos nessa visita, no espaço euclidiano, têm índices da curva de Hilbert próximos. Liu e Snoeyink (2005) utilizaram o algoritmo *counting sort* para a ordenação dos índices da curva de Hilbert, que é um algoritmo que pode ocorrer em tempo linear.

Em uma versão não publicada e gentilmente nos enviada pelo Professor Doutor J. Snoeyink, Snoeyink e Liu (2013) comparam a eficiência em se utilizar a ordem de quatro curvas de preenchimento de espaço em algoritmos incrementais. Essas curvas foram: de Gray (1953) e Hilbert (1891), ordenação em Z e a *ordenação pela linha principal*. Esses autores também testaram a inserção de pontos em ordem aleatória. Na ordenação Gray, utiliza-se um sistema binário numérico, em que a representação de dois valores sucessivos na ordem de Gray diferem em somente um *bit*. Na

ordenação pela linha principal, associam-se os pontos na primeira coluna a valores entre $\{1, 2, \cdots, 2^k\}$. Na i-ésima coluna, os pontos são ordenados de $\{(i\text{-}1)\times 2^k + 1, \cdots, i\cdots 2^k\}$.

Segundo Snoeyink e Liu (2013), as diferenças entre os tempos de execução com a utilização dessas curvas não foi impressionante. Os autores afirmaram ter escolhido a curva de Hilbert por conta da pequena vantagem obtida no que se refere ao tempo de execução ao utilizar a ordem dessa curva em relação às demais ordens de inserção de pontos.

Liu e Snoeyink (2005) compararam o algoritmo incremental utilizando a ordem da curva de Hilbert para a inserção de pontos com os algoritmos CGAL Devillers (1998), Boissonnat et al. (2002), QHull (CLARKSON, 1992), Barber, Dobkin e Huhdanpa (1996) e Pyramid (SHEWCHUK, 1998) O algoritmo de Liu e Snoeyink (2005) obteve os melhores custos computacionais dentre os algoritmos utilizados em testes.

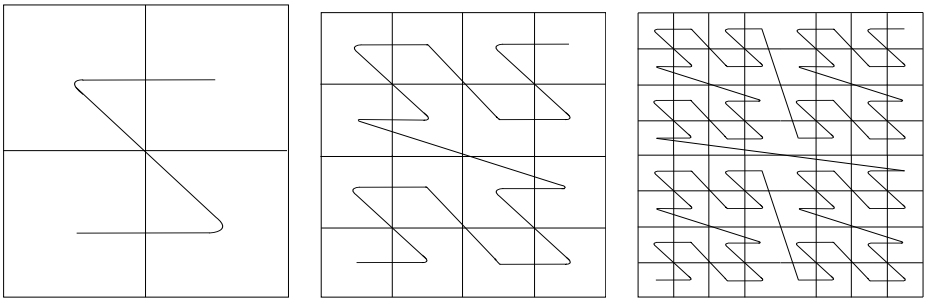
O algoritmo de Liu e Snoeyink (2005) representou uma mudança no conceito de desenvolvimento de algoritmos com comportamento linear para a geração de tesselações de Delaunay. Desde sua publicação, os algoritmos com comportamento linear para a geração da tesselação de Delaunay são, em sua maioria, incrementais e utilizam a inserção de pontos em uma ordem pré-determinada (OLIVEIRA; NOGUEIRA; TAVARES, 2014).

2.5.3 Curva de Lebesgue

A curva de preenchimento de espaço de Lebesgue foi proposta em Lebesgue (1904). Essa curva também é chamada de Z-order e de indexador de Morton (BADER, 2013). De maneira geomética, basicamente,

as aproximações unem os centros dos quadrados da esquerda para a direita, de baixo para cima e da esquerda para a direita novamente. Embora neste trabalho tenha-se utilizado o percurso da esquerda para a direita, de baixo para cima e da esquerda para a direita, variações desses percursos podem ser encontradas (por exemplo: da direita para a esquerda, de baixo para cima, da direita para a esquerda; de cima para baixo, da esquerda para a direita, de cima para baixo).

De maneira analítica, essa curva é associada ao conjunto de conjunto de Cantor. O conjunto de Cantor possui subconjuntos de valores de forma que [0,1] são definidos como limites para esse conjunto. Nesse conjunto, retira-se o terço do meio de cada um dos intervalos dos conjuntos criados na iteração anterior. Na Figura 12, pode-se observar a curva de Lebesgue bidimensional em três passos.



(a) Curva de Lebesgue com um nível de refinamento.

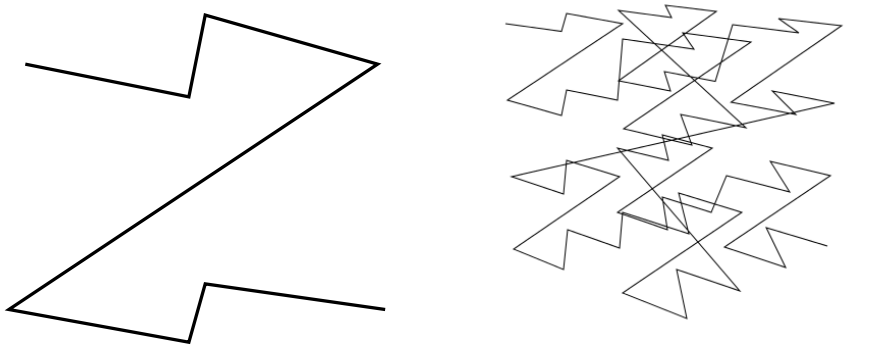
(b) Curva de Lebesgue com dois níveis de refinamento.

(c) Curva de Lebesgue com três níveis de refinamento.

Figura 12 Curva de Lebesgue bidimensional.

A versão tridimensional dessa curva é equivalente à versão bidimensional. Na versão tridimensional, o domínio é subdividido em oito subcubos. Para cada subcubo, visitam-se as subdivisões da mesma forma

que no domínio bidimensional; porém, subcubos cujas coordenadas em z são menores que a metade das coordenadas z do cubo original são visitados primeiro que os subcubos que contêm coordenadas z cujos valores são maiores que a metade do valor da coordenada z do cubo original. Na Figura 13, pode-se observar a curva de Lebesgue tridimensional, com um e com dois níveis de recursão.



(a) Curva de Lebesgue com um nível de refinamento.

(b) Curva de Lebesgue com dois níveis de refinamento.

Figura 13 Curva de Lebesgue tridimensional.

2.6 Esquema de indexação *H-indexing*

Um esquema bidimensional para indexação de malhas chamado *H-indexing* foi proposto por Niedermeier, Reinhardt e Sanders (2002). Esse esquema é baseado em uma variante bidimensional da curva de Sierpiński.

Niedermeier, Reinhardt e Sanders (2002) provaram que o *H-indexing* possui localidade melhor que os indexadores de Hilbert. Além disso, segundo os autores, há a conjectura de que o *H-indexing* é ótimo na preservação de localidade, dentre todos os indexadores de malha.

No esquema de indexação *H-indexing*, as coordenadas para $x(i)$ e $y(i)$, na i-ésima iteração, são definidas, recursivamente, da maneira apresentada a seguir. Considere que $l = k$-1, $g = 2^{2l-1}$ e $h = 2^{2l-3}$. Com isso, tem-se

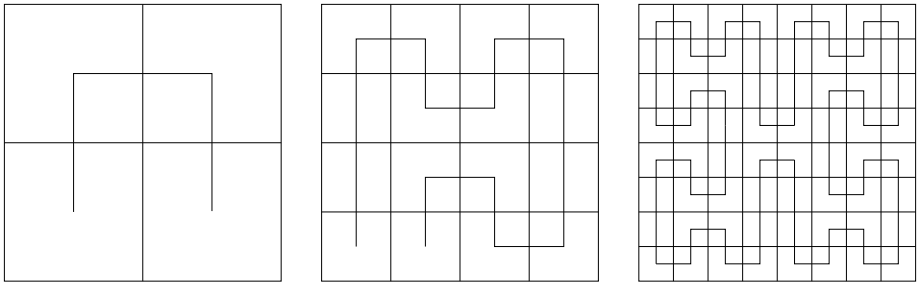
$$x(i) = \begin{cases} 2^k - 1 - x(i - 2^{2k-1}), \text{ se } i \geq 2^{2k-1} \\ 2^l + x(i - 3 \cdot 2^{2l-1}), \text{ se } 4 \cdot 2^{2l-1} \geq i \geq 3 \cdot 2^{2l-1} \\ 2^l - 1 - x(3 \cdot 2^{2l-1} - 1 - i), \text{ se } 3 \cdot 2^{2l-1} > i \geq 2 \cdot 2^{2l-1} \\ x(2^{2l} - 1 - i), \text{ se } 2 \cdot 2^{2l-1} > 1 \geq 2^{2l-1} \\ 0, \text{ se } i \leq 1 \end{cases}$$

e

$$y(i) = \begin{cases} 2^k - 1 - y(i - 2^{2k-1}), \text{ se } i \geq 2^{2k-1} \\ 2^l + y(i - 3 \cdot 2^{2l-1}), \text{ se } 4 \cdot 2^{2l-1} \geq i \geq 3 \cdot 2^{2l-1} \\ 2^l - y(3 \cdot 2^{2l-1} - 1 - i), \text{ se } 3 \cdot 2^{2l-1} > i \geq 2 \cdot 2^{2l-1}. \\ 2^{2l+1} - 1 - y(2^{2l} - 1 - i), \text{ se } 2 \cdot 2^{2l-1} > 1 \geq 2^{2l-1} \\ i, \text{ se } i \leq 1 \end{cases}$$

Na Figura 14, pode ser observada a construção da *H-indexing* em três passos. Nessa figura, os valores para k são 1, 2 e 3.

O esquema de indexação *H-indexing* foi proposto somente para indexações bidimensionais. Por esse motivo, para utilizar esse esquema em estruturas tridimensionais, optou-se por, inicialmente, ordenar o conjunto de pontos por suas coordenadas em z e, logo após, aplicar o esquema de indexação bidimensional em pontos com mesmas coordenadas em z.



(a) *H-indexing* com valor de k igual a um. (b) *H-indexing* com valor de k igual a dois. (c) *H-indexing* com valor de k igual a três.

Figura 14 Indexador *H-indexing*.

2.7 Ordem espiral

A ordenação em espiral de um conjunto de pontos foi proposta em Duff, Erisman e Reid (1976). Neste trabalho, na ordem espiral, o domínio é percorrido da esquerda para a direita, de baixo para cima, da direita para a esquerda e de cima para baixo. Esse trajeto ocorre novamente, de forma mais interna, até que todas as subdivisões do domínio sejam percorridas. Caso dois ou mais pontos estejam em uma mesma subdivisão, o esquema de indexação é aplicado novamente nessa subdivisão. Na Figura 15, pode ser observado um exemplo em que se percorre um conjunto de pontos em ordem espiral. Nesse exemplo, é utilizado um conjunto de cem pontos.

Assim como ocorreu ao utilizar o indexador *H-indexing*, para utilizar a ordem espiral em estruturas tridimensionais, optou-se, inicialmente, por ordenar o conjunto de pontos por suas coordenadas em z. Logo após, aplicou-se o esquema de ordenação em espiral bidimensional em pontos com mesmas coordenadas em z.

28 27 26 25 24 23 22 21 20 19
29 58 57 56 55 54 53 52 51 18
30 59 80 79 78 77 76 75 50 17
31 60 81 94 93 92 91 74 49 16
32 61 82 95 100 99 90 73 48 15
33 62 83 96 97 98 89 72 47 14
34 63 84 85 86 87 88 71 46 13
35 64 65 66 67 68 69 70 45 12
36 37 38 39 40 41 42 43 44 11
1 2 3 4 5 6 7 8 9 10

Figura 15 Ordem espiral para um conjunto de cem pontos.

2.8 Árvore rubro-negra

A árvore rubro-negra, proposta por Guibas e Sedgewick (1978), é aproximadamente balanceada, porque nenhum caminho da raiz até uma folha é duas vezes maior que outro (OLIVEIRA, 2011). Oliveira (2011) ainda explica que em uma árvore com algum tipo de balanceamento, pode-se mostrar que a altura máxima dessa árvore é limitada por um fator logarítmico.

As operações de inserção, exclusão, intercalação e divisão na árvore rubro-negra ocorrem em tempo $O(\lg\ n)$, em que n é o número de elementos da árvore. Cada nó interno contém uma chave e esse nó possui ligação para outros dois nós, que podem ou não ser externos. Nós externos não contêm chaves e têm ligações para ponteiros nulos. Além disso, cada nó possui uma cor, que pode ser vermelha ou preta. Esse esquema binário de cores permite que a árvore permaneça balanceada. Essa árvore também possui as seguintes propriedades:

- a raiz e as folhas são pretas;

- se um nó é vermelho, então, seus dois filhos são pretos;
- para cada nó preto, em todos os possíveis caminhos, desde esse nó até um nó folha, o número de nós pretos é o mesmo.

Na Figura 16, pode-se observar um exemplo de árvore rubro-negra para um conjunto de dezessete pontos.

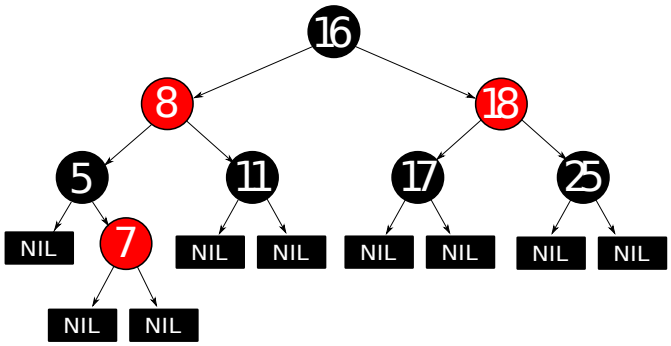


Figura 16 Árvore rubro-negra para um conjunto de oito pontos.

2.8.1 Inserção aleatória de pontos

O algoritmo incremental utilizando a inserção aleatória de pontos pode ser considerado a forma mais trivial de algoritmo incremental para a geração da tesselação de Delaunay. Ao inserir os pontos de maneira aleatória no domínio, não ocorre uma reordenação dos pontos de entrada.

Ou seja, nesse algoritmo, ao contrário do que ocorre com os demais algoritmos incrementais apresentados neste trabalho, não ocorre mudanças nos dados de entrada. Isso permite que o algoritmo incremental para a geração da tesselação de Delaunay seja mais facilmente implementado que os demais algoritmos incrementais.

3 METODOLOGIA E INFRAESTRUTURA

Nesta seção, apresentam-se a metodologia e os equipamentos necessários para o desenvolvimento deste trabalho. Na subseção 3.1, apresentam-se os equipamentos que foram utilizados. Na subseção 3.2 é apresentada a metodologia de pesquisa realizada. Na subseção 3.3, apresenta-se a metodologia de desenvolvimento utilizada neste trabalho. Na subseção 3.4, há uma explicação genérica para os algoritmos desenvolvidos. Na subseção 3.5, apresentam-se justificativas para as ordens de inserção de pontos selecionadas para implementação. Na subseção 3.6, é apresentada a estrutura de dados utilizada nos algoritmos bidimensionais desenvolvidos. Na subseção 3.7, apresenta-se a estrutura de dados utilizada nos algoritmos tridimensionais implementados. Finalmente, na subseção 3.8, é realizada uma descrição das ordens de inserção de pontos utilizadas neste trabalho.

3.1 Equipamentos

Para a realização dos testes, foram utilizadas três máquinas. Essas estações de trabalho são apresentadas a seguir.

- (M1) HP® Core™ i3 CPU 550 3.20 GHz;
- (M2) Dell Intel® Core™ i3-2120 CPU 3.30 GHz;
- (M3) Dell Intel® Core™ i3-2100 CPU 3.10 GHz (Intel; Santa Clara, CA, USA).

Para as três máquinas, o sistema operacional utilizado é o Ubuntu 14-04 LTS 64 bits, com *kernel* 3.13.0-39 generic e 4MB de mémória cache

na máquina M1, e *kernel* 3.13.0-43 generic com 3MB de memória cache nas máquinas M2 e M3. Em relação à memória principal, as máquinas M1 e M3 contêm 16GB e a máquina M2 contém 8GB de memória principal. A memória principal nas três máquinas é DDR3 1333MHz.

3.2 Metodologia de pesquisa

A metodologia de pesquisa utilizada neste trabalho iniciou-se com um estudo do problema de algoritmos para a geração da triangulação de Delaunay, do diagrama de Voronoi e da tetraedrização de Delaunay. Dentre os algoritmos estudados, foram verificados os algoritmos que ocorriam em tempo linear. O resultado obtido nesse estudo de algoritmos lineares para a geração das malhas de Delaunay e do diagrama de Voronoi pode ser verificado em Oliveira, Nogueira e Tavares (2014). Nesse trabalho de pesquisa de algoritmos lineares para a geração da tesselação de Delaunay e do diagrama de Voronoi, foram encontrados 34 algoritmos lineares que são utilizados para a construção dessas malhas.

Notou-se que, a partir da proposta do algoritmo BRIO, por Amenta, Choi e Rote (2003), os algoritmos lineares para a geração da tesselação de Delaunay e do diagrama de Voronoi, eram, em sua maioria, algoritmos incrementais. Além disso, grande parte dos algoritmos lineares propostos a partir do ano de 2003 foram utilizados para a tesselação de Delaunay. Por esse motivo, optou-se pela abordagem incremental, que foi utilizada para a construção da tesselação de Delaunay. As justificativas para a escolha das ordens de inserção de pontos que foram selecionadas para teste são apresentadas adiante, na subseção 3.5.

3.3 Metodologia de desenvolvimento

Para o alcance dos objetivos deste projeto, foram desenvolvidos códigos computacionais na linguagem C++, que realizavam a construção da tesselação de Delaunay em duas e em três dimensões. A biblioteca GNU MPFR com 256 *bits* de precisão foi utilizada para a obtenção de resultados mais precisos para os testes do circuncírculo, da circunsfera e de orientação.

Os algoritmos incrementais implementados utilizam as ordens das curvas de preenchimento de espaço de Hilbert e Lebesgue, a inserção aleatória de pontos, a ordem da árvore rubro-negra, com os percursos em ordem e em largura, a ordem espiral, a ordem determinada pelo indexador *H-indexing* e a ordem da árvore *cut-longest-edge kd-tree* para determinar a ordem em que ocorrerão as inserções de pontos na tesselação de Delaunay. Em estruturas bidimensionais, 6 distribuições de pontos foram utilizadas para teste. Em estruturas tridimensionais, 7 formas de distribuição de pontos foram utilizadas para teste. Essas distribuições de pontos, em estruturas bidimensionais e tridimensionais, ocorreram de maneira uniforme e não uniforme no intervalo unitário.

3.4 Explicação genérica para os algoritmos desenvolvidos

De maneira genérica, a seguir, serão apresentados os passos necessários para a construção da tesselação de Delaunay utilizando o algoritmo incremental utilizado neste trabalho. Para a geração da tesselação de Delaunay, em duas e em três dimensões, utilizou-se o algoritmo de Bowyer-Watson, que foi apresentado na subseção 2.3.

Nesse algoritmo, um ponto p, de um conjunto P, é inserido a cada iteração na malha de Delaunay. Independente da ordem utilizada, todos os pontos armazenados em P devem ser inseridos. Para todos os algoritmos desenvolvidos, exceto no algoritmo que realiza a inserção aleatória de pontos, inicialmente, ocorreu uma reordenação dos pontos de entrada. A reordenação dos pontos de entrada ocorreu utilizando-se as seguintes ordens: *cut-longest-edge kd-tree*, ordem das curvas de Hilbert e de Lebesgue, *H-indexing*, ordem em espiral e a árvore rubro-negra, com os percursos em ordem e em largura. A ordem resultante após a reordenação dos pontos de entrada é utilizada para determinar a ordem em que os pontos devem ser inseridos no domínio.

Para iniciar o algoritmo, constrói-se um supertriângulo (supertetraedro), que é um triângulo (tetraedro) que contém todos os pontos do domínio. A cada nova inserção, realiza-se uma busca para verificar um triângulo (tetraedro) t inicial cujo circuncírculo (circunsfera) contém p. Sabendo-se qual é o triângulo (tetraedro) t cujo circuncírculo (circunsfera) contém p, verifica-se se os circuncírculos (circunsferas) dos triângulos (tetraedros) adjacentes a t contêm p. Os triângulos (tetraedros) adjacentes a cada novo triângulo (tetraedro) cujo circuncírculo (circunsfera) contém p também devem ser verificados. A busca por triângulos (tetraedros) cujo circuncírculo (circunsfera) contém p ocorre até que a lista de triângulos (tetraedros) a serem verificados esteja vazia. Em todas as verificações, os testes de orientação, do circuncírculo (em estruturas bidimensionais) e da circunsfera (em estruturas tridimensionais), apresentados na subseção 2.1, são realizados.

Todos os triângulos (tetraedros) cujo circuncírculo (circunsfera)

contenha p devem ser destruídos. Os triângulos (tetraedros) na borda da cavidade formada após a destruição dos triângulos (tetraedros) devem ser unidos a p. A lista de adjacência dos triângulos (tetraedros) deve ser atualizada de acordo com as inserções e remoções de triângulos (tetraedros). O próximo ponto armazenado em P, após a ordenação, deve ser inserido na malha. Os passos apresentados anteriormente são realizados para esse novo ponto. O processo de inserção de pontos ocorre até que todos os pontos em P estejam inseridos no domínio.

Após a inserção de todos os pontos que estão em P, os vértices que pertencem ao supertriângulo (supertetraedro) devem ser removidos. Por esse motivo, todos os triângulos (tetraedros) que contêm vértices do supertriângulo (supertetraedro) devem ser excluídos. Dessa maneira, é gerada a triangulação (tetraedrização) de Delaunay. Essa abordagem de construção incremental da tesselação de Delaunay foi implementada neste trabalho.

3.5 Justificativa para a escolha das ordens de inserção

A seguir, são apresentadas justificativas para a escolha das ordens de inserção de pontos para os algoritmos incrementais que foram implementadas neste trabalho.

- *Cut-longest-edge kd-tree:* essa ordem de inserção de pontos apresenta custo computacional inferior ao custo computacional obtido ao utilizar a ordem da curva de Hilbert em algoritmos incrementais para a geração da tesselação de Delaunay. Como a inserção de pontos na ordem da curva de Hilbert era o provável estado da arte na geração da

tesselação de Delaunay e a inserção de pontos na ordem da *cut-longest-edge kd-tree* apresentou desempenho superior ao da curva de Hilbert; atualmente, esse algoritmo é o possível estado da arte na geração da tetraedrização de Delaunay. Por esse motivo, o algoritmo incremental utilizando a ordem do percurso em largura na *cut-longest-edge kd-tree* para a inserção dos pontos é utilizado.

- Curva de Hilbert: a inserção de pontos utilizando a ordem da curva de Hilbert foi implementada neste trabalho porque o algoritmo que utiliza a ordem dessa curva para a inserção de pontos em algoritmos incrementais era o possível estado da arte na geração de tesselações de Delaunay, até o ano de 2013. O uso da ordem dessa curva, para determinar a ordem em que os pontos eram inseridos no domínio, apresentou melhores custos computacionais que diversos algoritmos incrementais. Como exemplo, a inserção de pontos, na ordem da curva de Hilbert, apresentou custo computacional inferior ao apresentado pelo algoritmo BRIO, em experimentos realizados por Liu e Snoeyink (2005); também, a ordem da curva de Hilbert apresentou custos computacionais menores que os obtidos ao utilizar a ordem das curvas de Peano e de Sierpiński, em algoritmos incrementais, em testes realizados por Schrijvers, Bommel e Buchin (2012).

- Curva de Lebesgue: Schamberger e Wierum (2005) realizaram testes com as curvas de Hilbert, de Lebesgue, de Sierpiński e de $\beta\Omega$-*indexing*, proposta em Wierum (2002), para particionamento de malhas. Schamberger e Wierum (2005) notaram que, para os dados dispostos em uma estrutura espiral e com dados distribuídos de maneira uniforme, a curva de Lebesgue apresentou melhores

resultados que as demais curvas testadas. Com a curva de Lebesgue, foi necessária uma menor quantidade de partições do domínio, quando comparado às partições necessárias pelas demais curvas testadas por Schamberger e Wierum (2005). Embora a inserção de pontos em algoritmos incrementais utilizando a ordem da curva de Lebesgue tenha sido utilizada por Snoeyink e Liu (2013), os testes desses autores não ocorreram em estruturas bidimensionais e, os testes no 3D não ocorreram nas 7 distribuições de pontos utilizadas neste trabalho. Além disso, os testes de Snoeyink e Liu (2013) não ocorreram no cubo unitário.

- *H-indexing:* Niedermeier, Reinhardt e Sanders (2002) provaram que o *H-indexing* possui indexadores melhores que os indexadores de Hilbert. Além disso, Niedermeier, Reinhardt e Sanders (2002) apresentaram uma conjectura de que o *H-indexing* representa o ótimo na preservação da localidade. Como Niedermeier, Reinhardt e Sanders (2002) afirmam que os indexadores da *H-indexing* são melhores indexadores que os de Hilbert e a inserção de pontos utilizando a ordem da curva de Hilbert era o possível estado da arte na geração da tesselação de Delaunay até o ano de 2013, será verificado se a utilização desse indexador pode apresentar vantagens em relação às demais ordens de inserção de pontos que serão implementadas neste trabalho.

- Espiral: em Duff e Meurant (1989), estudaram-se 17 métodos de ordenação e os efeitos da ordenação das incógnitas de sistemas de equações lineares oriundas de discretizações de diferenças finitas sobre a convergência do método do gradiente conjugado pré-condicionado.

Duff e Meurant (1989) concluíram que ordenações locais, isto é, ordenações em que os nós vizinhos na malha base têm números que não estão muito distantes entre si, apresentam resultados melhores que as demais formas de ordenações. Porém, segundo os mesmos autores, a ordenação em espiral, que não é local, apresentou bons resultados para todos os conjuntos de testes.

- Árvore rubro-negra: Duff e Meurant (1989), no mesmo trabalho em que foram realizados os testes com a ordem espiral, descrito anteriormente, também mostraram que a ordem dada pela árvore rubro-negra era competitiva com as outras 16 sequências testadas. Esse esquema também foi avaliado por ser muito conhecido e na tentativa de encontrar uma árvore que apresentasse melhores resultados de tempo que os apresentados pela *cut-longest-edge kd-tree*.

- Inserção aleatória de pontos: essa inserção foi utilizada em testes por ser o esquema mais básico para que os pontos sejam inseridos. Além disso, a inserção aleatória dos pontos será utilizada para verificar o custo computacional de um algoritmo incremental sem que haja a fase de reordenação dos pontos.

As curvas de preenchimento de espaço de Peano e Sierpiński não foram implementadas neste trabalho. Isso ocorreu pois Schrijvers, Bommel e Buchin (2012) mostraram que o desempenho de algoritmos incrementais utilizando a sequência dada por essas curvas para a inserção dos pontos apresentaram custos computacionais superiores ao custo computacional ao se utilizar a ordem da curva de Hilbert.

3.6 Estruturas de dados para a triangulação de Delaunay

A estrutura de dados utilizada para armazenar os pontos que serão inseridos na triangulação é dada a seguir, na **struct coordenadas**, apresentada na Figura 17. Nessa figura, x e y são as coordenadas dos pontos e número é uma referência única a cada um dos pontos.

```
1 struct coordenadas{
2   double x, y;
3   int número;
4 };
```

Figura 17 Estrutura de dados utilizada para armazenamento das coordenadas dos pontos.

Cada ponto recebe um identificador, que é único, e que é armazenado na variável número. A variável número é útil para a atualização dos triângulos na malha, após as operações de inserção e remoção de um triângulo. Os valores armazenados em x, y e número permanecem inalterados, até o final da execução do algoritmo.

Sabe-se que cada triângulo da malha é formado por 3 pontos. Na estrutura de dados utilizada nas implementações e que é apresentada a seguir, na **struct triângulo**, dada na Figura 18, considera-se que cada triângulo é formado pelos pontos p1, p2 e p3. Os valores de p1, p2 e p3 são referências para os pontos já inseridos na malha. Os valores de p1, p2 e p3 são definidos de acordo com o valor armazenado na variável número, que pode ser acessado na **struct coordenadas**. Na **struct triângulo**, a variável númeroTriângulo é utilizada para armazenar um identificador único para cada triângulo. O identificador númeroTriângulo é utilizado para a manutenção da lista de triângulos adjacentes.

```
1 struct triângulo{
2   int p1, p2, p3, númeroTriângulo;
3 };
```

Figura 18 Estrutura de dados utilizada para armazenamento dos triângulos da malha.

Após uma operação de remoção, um triângulo inválido para a triangulação de Delaunay deve ser removido da lista de triângulos da malha. Após uma operação de inserção, um triângulo contendo os pontos p1, p2 e p3 é inserido na lista de triângulos da malha.

Para a utilização do algoritmo de Bowyer-Watson, é necessário que sejam armazenadas as relações de adjacências entre os triângulos da malha. Exceto para os triângulos da borda, cada triângulo da malha é adjacente a 3 triângulos. Os triângulos da borda são adjacentes a, no máximo, 2 triângulos.

A estrutura de dados utilizada para a manutenção das adjacências é a **struct listaDeAdjacência**, que é apresentada na Figura 19. As adjacências de um triângulo são armazenadas nas variáveis adj1, adj2 e adj3. Os valores de adj1, adj2 e adj3 são definidos de acordo com a variável númeroTriângulo, que é armazenada na **struct triângulo**.

```
1 struct listaDeAdjacência{
2   int adj1, adj2, adj3;
3 };
```

Figura 19 Estrutura de dados utilizada para armazenamento das adjacências.

A cada operação de exclusão de um triângulo, a lista de triângulos adjacentes ao triângulo removido deve ser atualizada. Ou

seja, o identificador do triângulo removido deve ser retirado da lista de adjacência de todos os triângulos adjacentes a ele. A cada operação de inserção, verificam-se novos triângulos adjacentes, ou seja, triângulos que compartilham arestas. Caso um triângulo t seja adjacente a t', t' é armazenado na lista de adjacências de t e t é armazenado na lista de adjacências de t'.

3.7 Estruturas de dados para a tetraedrização de Delaunay

As estruturas de dados utilizadas para a geração da tetraedrização de Delaunay são equivalentes às estruturas de dados utilizadas para a geração da triangulação de Delaunay. A estrutura de dados utilizada para armazenar os pontos que serão inseridos na tetraedrização é apresentada a seguir, na **struct coordenadas**, dada na Figura 20. Nessa estrutura, considere que x, y e z são as coordenadas dos pontos e número é uma referência única a cada um dos pontos.

```
1 struct coordenadas{
2   double x, y, z;
3   int número;
4 };
```

Figura 20 Estrutura de dados utilizada para armazenamento das coordenadas dos pontos.

Assim como ocorre na estrutura bidimensional, no algoritmo tridimensional, cada ponto recebe um identificador, que é único, e que é armazenado na variável número. A variável número é utilizada na atualização dos tetraedros na malha, após as operações de inserção e remoção de um tetraedro. Os valores armazenados em x, y, z e número

devem permanecer inalterados até que seja finalizada a execução do algoritmo.

Sabe-se que cada tetraedro da malha é formado por 4 pontos. Na estrutura de dados utilizada nas implementações, apresentada na **struct tetraedro**, dada na Figura 21, considera-se que cada tetraedro é formado pelos pontos p1, p2, p3 e p4. Os valores de p1, p2, p3 e p4 são referências a pontos já inseridos na malha e esses números são definidos de acordo com a variável número, utilizada na **struct coordenadas**. Também, cada tetraedro possui um identificador único, que é armazenado na variável númeroTetraedro. O identificador númeroTetraedro é utilizado para a manutenção da lista de tetraedros adjacentes.

```
1 struct tetraedro{
2   int p1, p2, p3, p4, númeroTetraedro;
3 };
```

Figura 21 Estrutura de dados utilizada para armazenamento dos tetraedros da malha.

Após uma operação de remoção, um tetraedro inválido para a malha de Delaunay é removido da lista de tetraedros. Após uma operação de inserção, um tetraedro contendo os pontos p1, p2, p3 e p4 é inserido na lista de tetraedros da malha.

Também, é necessário armazenar as relações de adjacências entre os tetraedros da malha. Exceto para tetraedros na borda da malha, que são adjacentes a, no máximo, 3 tetraedros, cada tetraedro é adjacente a 4 tetraedros. Para representar as adjacências entre os tetraedros, utiliza-se, neste trabalho, a **struct listaDeAdjacência**, apresentada na Figura 22. As adjacências de um tetraedro são armazenadas em adj1, adj2, adj3 e

adj4, cujos valores são definidos de acordo com a variável númeroTetraedro, armazenada na **struct tetraedro**.

```
1 struct listaDeAdjacência{
2   int adj1, adj2, adj3, adj4;
3 };
```

Figura 22 Estrutura de dados utilizada para armazenamento das adjacências.

A cada operação de remoção, a lista de tetraedros adjacentes ao tetraedro removido deve ser atualizada. Ou seja, o identificador do tetraedro removido deve ser retirado de todas as listas de adjacências dos tetraedros adjacentes a esse tetraedro invadido. A cada operação de inserção, verificam-se novos tetraedros adjacentes, ou seja, tetraedros que compartilham faces. Caso um tetraedro t seja adjacente a t', t' é armazenado na lista de adjacências de t e t é armazenado na lista de adjacências de t'.

3.8 Descrição das ordens de inserção implementadas

Nas subseções a seguir, descrevem-se as ordens de inserção de pontos implementadas neste trabalho. Essas descrições são apresentadas para as ordens de inserção de pontos em estruturas bidimensionais e tridimensionais.

3.8.1 *Cut-longest-edge kd-tree*

No algoritmo da *cut-longest-edge kd-tree*, as coordenadas x, y e z (em estruturas tridimensionais) dos pontos são, inicialmente, valores inteiros. Essas coordenadas possuem valores inteiros pois os algoritmos de ordenação

são mais eficientes para a ordenação de valores inteiros. Após a construção da *cut-longest-edge kd-tree*, essas coordenadas são convertidas para valores entre 0 e 1. Nesse algoritmo, utilizou-se a estrutura **Ponto**, apresentada na Figura 23. Considere que, x, y e z, são, respectivamente, as coordenadas x, y e z de um ponto.

```
1 Ponto(int x, int y, int z){
2   coordenadas[0] = x;
3   coordenadas[1] = y;
4   coordenadas[2] = z;
5 }
6 int coordenadas[3];
```

Figura 23 Estrutura de dados utilizada para armazenamento dos pontos na *cut-longest-edge kd-tree*.

Na **struct RefDimensao**, apresentada na Figura 24, armazenam-se o maior e o menor valor das coordenadas x, y e z (armazenados, respectivamente, nas variáveis maior e menor), a mediana do intervalo (armazenada na variável mediana) e o intervalo que contém a maior diferença entre as coordenadas x, y e z (armazenado na variável dimensão). Essa estrutura é apresentada a seguir.

```
1 struct RefDimensao{
2   int maior, menor, mediana, dimensao;
3 } ;
```

Figura 24 Estrutura de dados auxiliar utilizada na *cut-longest-edge kd-tree*.

Para cada nó inserido na árvore, são armazenadas informações sobre os nós filhos. Além disso, são armazenadas as coordenadas x, y e z, que estão armazenadas na estrutura **Ponto** e se o corte foi realizado em x, y ou z. Essas informações são armazenadas na estrutura **No**, que é apresentada na

Figura 25.

```
1 No(int dimensao, Ponto valor){
2   esq = dir = NULL; //nós filhos vazios
3   this->valor = valor; //coordenadas do ponto
4   this->dimensao = dimensao; //eixo em que ocorre a
        subdivisão do domínio
5 }
```

Figura 25 Estrutura para armazenamento dos nós da *cut-longest-edge kd-tree*.

As duas funções principais desse algoritmo são apresentadas nas Figuras 26 e 27. Na Figura 26, apresenta-se o algoritmo em que ocorre, efetivamente, a construção da *cut-longest-edge kd-tree*. Nesse algoritmo, o valor inicial das variáveis inf e sup são, respectivamente, 0 e o número total de pontos da *cut-longest edge kd-tree*. Já na variável P, são armazenados os pontos utilizados para a construção da *cut-longest-edge kd-tree* na iteração corrente.

A função obterMaiorDimensao, apresentada na Figura 27, é uma função auxiliar para o algoritmo apresentado na Figura 26. No algoritmo apresentado na Figura 27, os índices iniciais (inf) e finais (sup) e o conjunto de pontos P são passados pelo algoritmo apresentado na Figura 26.

No algoritmo apresentado na Figura 26, os pontos são ordenados utilizando o algoritmo *bucket sort*. O algoritmo *bucket sort* é um algoritmo de ordenação em que o conjunto de pontos inicial é subdividido em um número finito de recipientes (*buckets*). Cada recipiente é então ordenado individualmente.

Neste trabalho, os pontos foram inseridos de acordo com o percurso em largura na *cut-longest-edge kd-tree*. Ao realizar o percurso em largura,

```
    Entrada: inteiros: inf, sup, dimensao;
             conjunto P de pontos tipo Ponto.
    Saída: Ponto v a ser inserido na cut-longest-edge kd-tree.
 1  início
 2      se (inf=sup) então retorna NULL;
 3      se (sup-inf=1) então retorna No(P[inf], dimensao);
 4      RefDimensao dCorte←obterMaiorDimensao(inf, sup, P);
            // verificação do intervalo de maior diferença
            entre as coordenadas; P será ordenado se
            dCorte.dimensao é diferente de dimensao (que é
            o intervalo de maior diferença do nó pai)
 5      se (dCorte.dimensao≠dimensao) então
 6          se (dCorte.dimensao=0) então ordena P pelas
            coordenadas x dos pontos;
 7          senão
 8              se (dCorte.dimensao=1) então ordena P pelas
                coordenadas y dos pontos;
 9              senão ordena P pelas coordenadas z dos pontos;
10          fim se
11      fim se
12      Ponto pMed←P[inf+dCorte.mediana];
13      v ← No(dCorte.dimensao, pMed); // Nó raiz de uma
            subárvore, dado pela mediana do intervalo
        // chamadas recursivas para os pontos à esquerda e
            à direita de pMed, respectivamente
14      v->esq ← constróiCutLongestEdgeKDTree (inf, inf +
        dCorte.mediana, dCorte.dimensao);
15      v->dir ← constróiCutLongestEdgeKDTree (inf +
        dCorte.mediana+1, sup, dCorte.dimensao);
16      retorna v;
17  fim
```

Figura 26 constróiCutLongestEdgeKDTree.

visita-se, inicialmente, o nó raiz da árvore. Logo após, visitam-se os nós filhos *f1* e *f2*, à esquerda e à direita, respectivamente, desse nó raiz. Logo após, cada um dos filhos de *f1* e *f2* são visitados. Esse processo ocorre até que todos os nós da árvore sejam percorridos. Ao utilizar o percurso em largura, todos os nós que estão em um mesmo nível são visitados de forma que, somente após a visita de todos os nós de um mesmo nível, possam ser percorridos nós do nível inferior da árvore.

```
    Entrada: inteiros: inf, sup;
             conjunto P de pontos do tipo Ponto.
    Saída: dCorte, que é uma variável do tipo RefDimensao.
 1  início
 2      inteiros: maior[3], menor[3], med[3], maiorI, maiorD, i,d←0;
 3      enquanto (d<3) faça
            // verificação para todos os pontos em P, nas
               coordenadas x (d=0), y (d=1) e z (d=2)
 4          maior[d]←-∞; menor[d]← ∞; i←inf;
 5          enquanto (i<sup) faça
                // verifica se é necessário atualizar os
                   valores de menor[d] e maior[d]
 6              se (P[i].coordenadas[d]>maior[d]) então
 7              maior[d] ← P[i].coordenadas[d];
 8              se (P[i].coordenadas[d]<menor[d]) então
 9              menor[d] ← P[i].coordenadas[d];
10              i←i+1;
11          fim enquanto
12          se (maior[d]-menor[d]>maiorI) então
                // atualização do intervalo de maior
                   diferença entre as coordenadas
13              maiorI←maior[d]-menor[d]; maiorD←d;
14          fim se
15          med[d]←(sup-inf)/2; d←d+1;
16      fim enquanto
17      RefDimensao dCorte← ∅;
18      dCorte.maior←maior[maiorD];
        dCorte.menor←menor[maiorD];
19      dCorte.mediana←med[maiorD]; dCorte.dimensao←maiorD;
20      retorna dCorte;
21  fim
```

Figura 27 obterMaiorDimensao.

3.8.2 Curvas de Hilbert e de Lebesgue

Nas implementações bidimensionais, ao se utilizar a ordem das curvas de Hilbert ou de Lebesgue para definir a ordem em que os pontos devem ser inseridos, o domínio é dividido em 2^{2k} subdivisões, de forma que k é um número grande o suficiente para que cada subdivisão contenha uma quantidade pequena de pontos. Caso uma subdivisão contenha mais de um ponto, essa subdivisão é dividida, novamente, em 4 subquadrados, até que cada subdivisão contenha apenas um ponto.

Quando dois ou mais pontos, em estruturas bidimensionais, caem em uma mesma subdivisão, é necessário realizar uma nova subdivisão do intervalo. Para isso, utiliza-se a estrutura de dados *quadtree*, em que são armazenadas referências para os pontos que estão em uma mesma subdivisão. A *quadtree* é uma árvore em que cada nó contém, no máximo, quatro nós filhos. Na implementação utilizada neste trabalho, os valores das coordenadas x e y, que estão entre 0 e 1, são convertidos para valores inteiros. Para esses valores inteiros, o índice das curvas de Hilbert ou de Lebesgue correspondente é calculado.

A ordenação dos pontos de acordo com as curvas de Hilbert e de Lebesgue, em estruturas tridimensionais, ocorre de maneira similar à que ocorre em estruturas bidimensionais. Em estruturas tridimensionais, o domínio é dividido em 2^{3k} subdivisões, de forma que k é um número grande o suficiente para que cada subdivisão contenha uma quantidade pequena de pontos. Em algoritmos tridimensionais, quando dois ou mais pontos caem em uma mesma subdivisão, utiliza-se a estrutura de dados *octree*, estrutura em que são armazenadas referências para os pontos que estão em uma mesma subdivisão. A *octree* é uma árvore em que cada nó contém, no máximo, oito nós filhos. Na implementação utilizada neste trabalho, os valores das coordenadas x, y e z, que estão entre 0 e 1, são convertidos para valores inteiros. Para esses valores inteiros, o índice das curvas de Hilbert ou de Lebesgue correspondente é calculado.

Conforme descrito anteriormente, existem diversas formas de geração da curva de Hilbert tridimensional. A versão da curva de Hilbert tridimensional utilizada neste trabalho é apresentada em Robaina et al. (2010).

Para ordenar os índices das subdivisões, utilizou-se o algoritmo *radix-sort*. O *radix-sort* é um algoritmo em que a ordenação dos dados ocorre pelo processamento dos dígitos de maneira individual. Esse algoritmo ocorre, no pior caso, em tempo $O(nk)$, em que n é o número de pontos e k é o número máximo de dígitos dos números que estão sendo ordenados.

3.8.3 *H-indexing*

Neste algoritmo, o domínio é dividido em 2^{2k} subdivisões e a cada subdivisão é associado um índice do *H-indexing*. Caso dois ou mais pontos estejam em uma mesma subdivisão, o esquema de indexação utilizando a ordem dada pelo *H-indexing* é aplicado novamente nessa subdivisão. Em testes, observou-se que os custos computacionais obtidos ao utilizar-se valores menores ou iguais a 4096×4096 para o número de subdivisões do domínio, a indexação dos pontos ocorria com custo computacional mais baixo que o custo computacional para o número de subdivisões utilizado neste trabalho. Porém, ao se utilizar esses valores, os custos computacionais obtidos para a construção da triangulação de Delaunay eram superiores aos custos computacionais com valores acima de 4096×4096 subdivisões. Também, ao utilizar valores menores ou iguais a 4096×4096, os custos computacionais obtidos em diferentes execuções de uma mesma instância variavam de maneira considerável. Por esse motivo, adotou-se uma subdivisão inicial da malha de 8192×8192 em estruturas bidimensionais.

Ao utilizar a ordem do indexador *H-indexing* em domínios tridimensionais, os pontos são ordenados, inicialmente, por suas coordenadas em z. Logo após, pontos que possuem a mesma coordenada z

são ordenados utilizando a ordem do indexador *H-indexing*. Caso dois ou mais pontos estejam em uma mesma subdivisão, o esquema de indexação utilizando a ordem *H-indexing* é aplicado novamente na subdivisão. Para todas as ordenações, utilizou-se o algoritmo *radix-sort*.

3.8.4 Ordem espiral

Neste algoritmo, conforme descrito anteriormente, o domínio é percorrido da esquerda para a direita, de baixo para cima, da direita para a esquerda, de cima para baixo. Esse caminho é realizado até que todas as subdivisões do domínio sejam percorridas. Caso dois ou mais pontos estejam em uma mesma subdivisão, o esquema de indexação utilizando a ordem espiral é aplicado novamente na subdivisão.

Em esquemas bidimensionais, notou-se que, ao utilizar valores altos, geralmente acima de 4000×4000, para a subdivisão do domínio, os custos computacionais para a construção da triangulação de Delaunay eram altos. Além disso, valores acima de 7000×7000 subdivisões do domínio tornavam inviável o uso da ordem espiral para a construção da triangulação de Delaunay. Em testes, em geral, o valor que apresentou melhores custos computacionais foi o de 2000×2000 subdivisões. Esse número de subdivisões do domínio foi utilizado para todos os testes realizados em estruturas bidimensionais.

Ao utilizar a ordem espiral em domínios tridimensionais, os pontos são ordenados, inicialmente, por suas coordenadas em z. Logo após, pontos de mesma coordenada z são ordenados utilizando a ordem espiral. Caso dois ou mais pontos estejam em uma mesma subdivisão, o esquema de indexação

utilizando a ordem espiral é aplicado novamente na subdivisão. Para todas as ordenações de pontos, utilizou-se o algoritmo *radix-sort*.

3.8.5 Árvore rubro-negra

Na árvore rubro-negra, as cores vermelha e preta são utilizadas. Por esse motivo, neste trabalho, foi necessária a utilização das estruturas de dados apresentadas nas Figuras 28 e 29.

```
1 enum corDoNó{
2     VERMELHO, PRETO //cores possíveis de um nó
3 };
```

Figura 28 Estrutura auxiliar utilizada na construção da árvore rubro-negra.

Cada nó inserido na árvore contém os dados conforme apresentado na **struct rubroNegraNó**. Nessa estrutura, x, y e z são, respectivamente, as coordenadas x, y e z de um ponto, cor é a cor do nó inserido e esq e dir são ponteiros para os nós filhos do nó. A **struct rubroNegraNó** é apresentada na Figura 29.

```
1 struct rubroNegraNó {
2     double x, y, z; //coordenadas x, y e z de um ponto
3     int cor; //cor do nó
4     struct corDoNó *esq, *dir; //nós filhos
5 };
```

Figura 29 Estrutura de dados utilizada para a construção da árvore rubro-negra.

A variável cor é inicializada como vermelha. Caso ocorra, após uma operação de inserção na árvore rubro-negra, de um nó vermelho ter um nó filho vermelho ou o número de nós pretos de qualquer caminho percorrido ser diferente, ocorre o balanceamento dos nós da árvore rubro-negra de forma

que as propriedades da árvore rubro-negra sejam mantidas. Para detalhes de como ocorrem esses balanceamentos, veja Oliveira (2011).

Para a inserção de pontos na tesselação de Delaunay utilizando a árvore rubro-negra, foram utilizadas duas formas de percurso: em ordem e em largura. No percurso em ordem, visita-se, inicialmente o nó filho à esquerda, logo após, o nó raiz e, finalmente, o nó filho à direita. Já no percurso em largura, visitam-se os nós da árvore de acordo com os níveis em que os nós encontram-se na árvore. Inicialmente, o nó raiz da árvore é visitado. Logo após, visitam-se os nós filhos *f1* e *f2*, à esquerda e à direita do nó raiz, respectivamente. Então, cada um dos filhos de *f1* e *f2* devem ser visitados. Esse processo ocorre até que todos os nós sejam percorridos.

3.8.6 Inserção aleatória de pontos

O algoritmo incremental utilizando a inserção aleatória de pontos não utiliza estrutura de dados extra para o armazenamento dos dados de entrada. Isso ocorre pois não é necessário reordenar os dados de entrada.

De maneira geral, os pontos são inseridos no domínio sem que ocorra uma reordenação dos dados de entrada. Por não realizar a reordenação dos pontos, o algoritmo incremental utilizando a inserção aleatória de pontos foi o algoritmo de mais fácil implementação dentre os algoritmos incrementais utilizados neste trabalho.

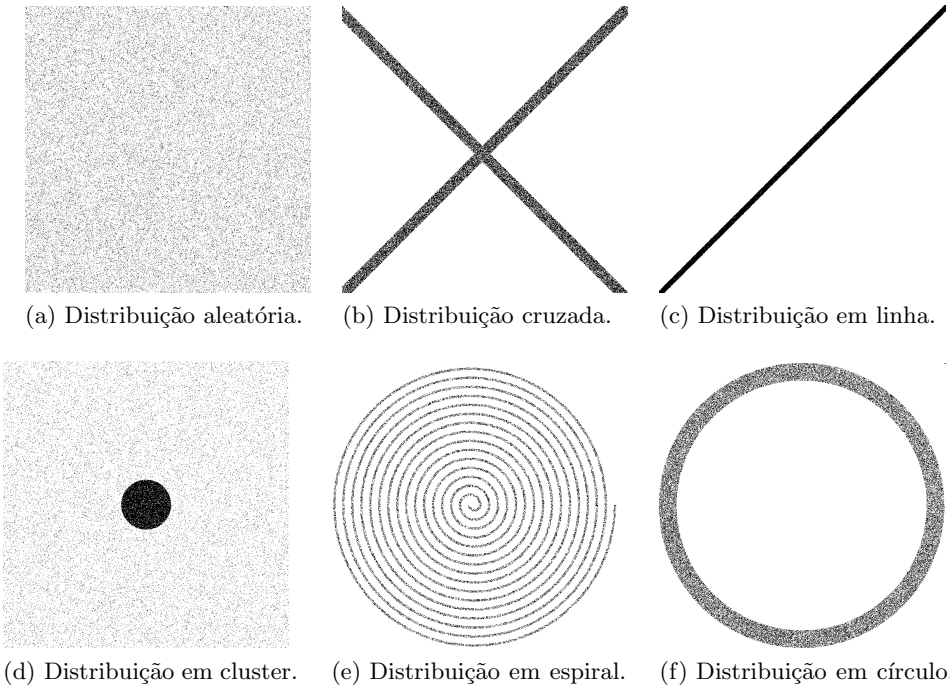
4 RESULTADOS PARA A TRIANGULAÇÃO DE DELAUNAY E ANÁLISES

Nesta seção, são apresentadas as distribuições de pontos utilizadas em testes e os resultados de custo computacional e utilização de memória obtidos nas 8 sequências de inserção de pontos nos algoritmos incrementais. Na subseção 4.1, são apresentadas as 6 distribuições de pontos utilizadas em testes. Na subseção 4.2, são apresentados os custos computacionais e ocupações de memória nas diferentes ordens de inserção de pontos, exceto pela ordem dada pela *cut-longest-edge kd-tree*, utilizando três diferentes abordagens para a inserção de pontos. Na subseção 4.3, são apresentadas as médias de custo computacional, ocupação de memória, desvio padrão e coeficiente de variação nas ordens de inserção de pontos selecionadas para teste. Finalmente, na seção 4.4, conclusões sobre os testes são apresentadas.

4.1 Distribuições de pontos utilizadas para teste

Nos testes bidimensionais, foram utilizadas 6 distribuições de pontos, que foram: aleatória, cruzada, em linha, em cluster, em círculo e em espiral. Exemplos para essas distribuições são apresentados na Figura 30.

Os conjuntos de pontos utilizados contêm entre 25000 e 1000000 de pontos e as coordenadas dos pontos estão no intervalo unitário. Uma vez que os testes ocorreram no intervalo unitário, a biblioteca de precisão GNU MPFR com 256 *bits* de precisão foi utilizada para tornar possível obter uma maior precisão nos testes de posicionamento e do circuncírculo.



(a) Distribuição aleatória. (b) Distribuição cruzada. (c) Distribuição em linha.

(d) Distribuição em cluster. (e) Distribuição em espiral. (f) Distribuição em círculo.

Figura 30 Distribuições de pontos utilizadas em teste. Nos exemplos, os conjuntos possuem 50000 pontos.

4.2 Resultados empíricos utilizando 3 formas de busca de pontos

Exceto no algoritmo incremental utilizando a ordem da *cut-longest-edge kd-tree*, foram realizadas 3 formas de busca por um triângulo cujo circuncírculo contém o ponto mais recentemente inserido na triangulação, que foram:

1. *hist_triang*: triângulos são verificados na ordem inversa em que eles foram inseridos na triangulação.

2. *ult_triang:* realiza-se uma busca em largura que é iniciada pelo último

triângulo criado.

3. *triang_inc*: k triângulos incidentes ao último ponto p inserido na triangulação são verificados. Se os circuncírculos dos triângulo verificados não contiverem p, triângulos adjacentes aos k triângulos também são verificados, a partir do triângulo menos recentemente verificado.

Essa busca ocorre até que um circuncírculo de um triângulo contendo p seja encontrado. Para o algoritmo incremental utilizando a inserção de pontos na ordem dada pela *cut-longest-edge kd-tree*, a busca pelo circuncírculo contendo o ponto p é realizada pelos k triângulos incidentes ao nó pai de p na *cut-longest-edge kd-tree*.

Os resultados obtidos nas 7 ordens de inserção, utilizando as 3 formas de busca pelo ponto mais recentemente inserido e nos 6 conjuntos de pontos são apresentados nas tabelas entre 1 e 7. O menor custo computacional em cada uma das instâncias foi colocado em negrito. Nessas tabelas, considere que o símbolo "-" indica que a execução foi abortada. As execuções foram abortadas após 10 minutos de execução do algoritmo.

Tabela 1 Resultados de custo computacional (em segundos) e ocupação de memória (em *megabytes*) utilizando a ordem da curva de Hilbert.

Curva de Hilbert							
Número de pontos	Resultados	*hist_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	**1,54**	1,86	1,87	**1,63**	**1,79**	**1,69**
	Memória	25,04	25,05	29,76	24,85	24,71	24,48
50000	Tempo	3,22	3,86	3,87	3,56	**3,59**	3,55
	Memória	32,19	31,23	32,71	32,75	30,66	30,68
75000	Tempo	**4,66**	5,97	6,78	5,50	6,20	5,50
	Memória	39,54	41,02	41,48	41,69	39,26	38,80
100000	Tempo	**6,25**	7,82	8,57	**6,66**	**7,35**	**7,27**
	Memória	46,94	46,15	49,38	48,78	44,15	47,14
250000	Tempo	**16,17**	21,75	26,16	18,35	**18,86**	19,09
	Memória	88,27	91,55	106,43	96,84	89,00	89,89
500000	Tempo	**32,57**	54,95	78,75	**34,96**	**37,36**	39,04
	Memória	161,19	182,57	217,91	184,25	169,92	167,41
750000	Tempo	**49,30**	95,27	129,09	52,52	56,68	61,10
	Memória	245,97	275,06	370,75	283,28	253,83	253,04
1000000	Tempo	72,17	149,57	214,36	**69,36**	**73,89**	84,85
	Memória	314,64	380,53	569,12	362,35	351,61	353,71
Número de pontos	Resultados	*ult_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	1,59	1,78	**1,60**	1,64	1,80	1,71
	Memória	24,01	24,28	24,57	24,32	24,33	24,34
50000	Tempo	**3,20**	3,62	**3,29**	**3,34**	3,79	**3,46**
	Memória	29,87	30,58	31,61	30,43	30,33	30,35
75000	Tempo	4,83	5,81	**5,09**	**5,20**	**5,55**	**5,27**
	Memória	35,89	37,17	39,16	37,10	37,27	37,33
100000	Tempo	6,57	**7,16**	**6,96**	6,77	7,52	7,52
	Memória	41,83	44,63	47,32	44,94	42,92	43,53
250000	Tempo	17,20	**19,09**	**19,07**	**17,24**	19,02	18,82
	Memória	79,24	88,00	103,79	87,25	85,03	84,93
500000	Tempo	33,79	**40,79**	**40,74**	35,39	38,01	**37,04**
	Memória	142,17	173,56	213,08	165,55	159,84	161,64
750000	Tempo	50,67	**67,69**	**65,59**	**52,43**	**56,29**	**55,67**
	Memória	208,23	268,55	356,79	245,77	244,09	247,14
1000000	Tempo	**68,51**	**95,03**	**94,47**	70,88	74,71	**75,29**
	Memória	276,39	369,10	559,02	325,64	332,15	336,52
Número de pontos	Resultados	*triang_inc*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	1,59	**1,74**	2,19	1,82	1,87	**1,69**
	Memória	26,33	26,60	26,89	26,64	26,40	26,41
50000	Tempo	3,21	**3,59**	3,37	3,39	3,76	3,52
	Memória	34,51	35,15	36,07	35,00	34,71	34,80
75000	Tempo	5,09	**5,72**	5,35	5,35	5,58	5,29
	Memória	42,52	43,88	46,12	43,81	43,39	43,96
100000	Tempo	6,59	7,76	7,63	7,54	7,69	7,45
	Memória	51,36	54,30	56,60	54,60	51,95	52,99
250000	Tempo	17,35	20,03	21,60	18,43	19,02	**18,68**
	Memória	101,84	111,71	126,64	110,07	108,31	107,93
500000	Tempo	35,68	43,17	43,13	37,72	40,35	37,96
	Memória	187,58	221,91	259,26	210,69	205,78	207,13
750000	Tempo	55,54	72,08	69,80	57,13	60,91	57,93
	Memória	276,61	339,77	430,74	313,70	312,09	314,94
1000000	Tempo	73,07	104,50	97,95	75,70	80,44	76,03
	Memória	368,30	465,18	655,60	417,30	424,64	427,13

Tabela 2 Resultados de custo computacional (em segundos) e ocupação de memória (em *megabytes*) utilizando a ordem da curva de Lebesgue.

Curva de Lebesgue							
Número de pontos	Resultados	*hist_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	**2,27**	**2,10**	**1,70**	**2,36**	**2,50**	**2,08**
	Memória	24,90	25,46	25,49	25,22	24,98	25,23
50000	Tempo	**4,59**	**4,43**	**3,65**	**4,99**	**5,13**	**4,59**
	Memória	31,60	32,51	33,40	32,41	31,86	31,82
75000	Tempo	**7,04**	**7,68**	**5,50**	**7,60**	8,82	**7,22**
	Memória	38,34	39,64	40,32	39,73	39,13	38,99
100000	Tempo	**9,53**	**9,88**	**7,67**	**10,53**	**10,72**	**9,67**
	Memória	46,26	47,64	50,68	48,92	46,10	46,97
250000	Tempo	**25,43**	**26,20**	**22,13**	**28,23**	**27,68**	**26,76**
	Memória	86,69	98,37	109,88	94,20	91,94	94,38
500000	Tempo	**53,39**	**57,19**	**47,82**	**57,46**	**57,97**	**55,00**
	Memória	159,65	193,07	225,12	180,40	176,24	176,93
750000	Tempo	**79,30**	**92,82**	**79,40**	**90,34**	**88,92**	**85,10**
	Memória	233,46	290,16	372,94	275,15	277,37	272,16
1000000	Tempo	**113,44**	**132,11**	**107,49**	**116,13**	**123,38**	**113,54**
	Memória	313,41	400,87	580,65	359,87	364,80	366,31
Número de pontos	Resultados	*ult_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	2,54	2,23	1,82	2,72	2,59	2,37
	Memória	24,65	24,69	24,98	24,96	24,72	30,13
50000	Tempo	5,41	4,85	4,07	6,25	5,26	5,29
	Memória	31,08	31,55	32,37	31,83	31,35	31,30
75000	Tempo	8,65	7,91	6,76	9,98	8,60	8,55
	Memória	37,87	39,29	40,49	39,22	38,17	38,87
100000	Tempo	11,93	10,97	8,53	12,80	11,78	11,46
	Memória	44,24	45,98	49,14	47,74	45,65	44,91
250000	Tempo	34,70	30,82	25,48	36,93	30,37	33,95
	Memória	85,53	92,58	106,87	93,69	89,50	90,46
500000	Tempo	74,91	72,62	57,96	80,32	63,21	76,13
	Memória	156,23	183,04	218,53	178,51	170,16	171,74
750000	Tempo	123,60	122,25	98,34	124,21	97,68	124,74
	Memória	227,37	278,77	362,97	265,46	258,96	263,34
1000000	Tempo	164,59	180,00	147,41	175,08	132,35	180,16
	Memória	301,72	380,85	566,86	353,85	352,85	355,36
Número de pontos	Resultados	*triang_inc*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	2,62	2,37	**1,70**	2,70	2,53	2,27
	Memória	26,96	27,01	27,30	27,28	27,04	27,04
50000	Tempo	6,11	5,07	3,84	6,49	5,20	5,20
	Memória	35,47	36,13	36,94	36,53	35,99	35,76
75000	Tempo	9,51	8,20	6,15	9,51	**8,38**	8,00
	Memória	45,00	45,50	47,26	45,73	45,32	45,25
100000	Tempo	12,31	11,69	8,42	12,86	11,91	11,26
	Memória	53,25	55,53	58,23	56,44	55,43	54,23
250000	Tempo	36,66	34,06	25,51	38,35	30,26	34,18
	Memória	108,63	116,68	130,04	116,18	112,68	112,79
500000	Tempo	80,49	84,36	63,15	85,22	63,21	79,15
	Memória	199,39	233,31	265,21	223,96	170,16	217,83
750000	Tempo	126,50	144,76	109,67	133,91	105,97	131,03
	Memória	293,28	357,65	438,04	331,06	329,90	330,16
1000000	Tempo	183,01	219,27	165,29	187,74	142,84	184,72
	Memória	390,57	486,85	667,78	443,27	448,98	449,48

Tabela 3 Resultados de custo computacional (em segundos) e ocupação de memória (em *megabytes*) utilizando a ordem da *H-indexing*.

		H-indexing					
Número de pontos	Resultados	*hist_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	36,89	36,53	**35,22**	**34,78**	37,09	**36,52**
	Memória	1033,70	1033,16	1033,80	1033,79	1033,54	1033,54
50000	Tempo	38,57	39,34	**36,48**	**36,56**	**38,97**	**38,28**
	Memória	1041,12	1039,50	1041,04	1041,27	1041,02	1040,96
75000	Tempo	**40,31**	41,41	**37,55**	**38,42**	**40,25**	**39,99**
	Memória	1048,62	1048,07	1047,59	1049,36	1047,54	1047,28
100000	Tempo	**41,61**	43,42	**39,52**	**39,58**	**42,10**	**41,75**
	Memória	1056,74	1054,21	1057,13	1056,78	1056,30	1056,29
250000	Tempo	**51,99**	65,13	**49,70**	52,26	**54,57**	**53,17**
	Memória	1100,86	1104,52	1098,89	1099,73	1099,63	1099,75
500000	Tempo	73,40	120,26	**69,99**	231,82	**73,79**	**73,23**
	Memória	1171,30	1176,72	1171,35	1172,16	1168,93	1170,36
750000	Tempo	**89,37**	210,00	**92,02**	335,66	98,57	**95,08**
	Memória	1253,37	1232,17	1254,00	1245,25	1248,50	1248,77
1000000	Tempo	109,38	325,62	**115,61**	584,46	**116,04**	**116,69**
	Memória	1318,89	1329,79	1320,64	1319,05	1312,69	1310,18
Número de pontos	Resultados	*ult_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	**36,70**	**36,52**	35,40	34,99	37,43	36,76
	Memória	1032,41	1032,50	1032,51	1032,50	1032,51	1032,51
50000	Tempo	**38,55**	38,65	36,55	36,87	39,24	39,55
	Memória	1038,80	1038,90	1038,90	1038,88	1038,90	1038,89
75000	Tempo	40,61	40,70	38,29	38,68	40,93	40,40
	Memória	1045,13	1045,23	1045,17	1045,63	1045,22	1046,04
100000	Tempo	41,95	**42,31**	40,59	40,52	43,55	42,61
	Memória	1051,47	1052,34	1051,50	1051,62	1052,48	1051,45
250000	Tempo	53,75	**53,34**	51,32	**52,08**	55,36	54,36
	Memória	1090,64	1089,68	1090,84	1089,91	1089,32	1089,96
500000	Tempo	**70,86**	**73,03**	70,66	**73,29**	77,03	74,85
	Memória	1152,98	1154,02	1153,57	1151,48	1153,21	1152,43
750000	Tempo	90,16	**96,03**	92,86	**95,93**	**97,91**	97,30
	Memória	1214,90	1216,35	1215,55	1212,10	1213,59	1214,55
1000000	Tempo	**106,86**	**118,64**	120,68	**119,88**	118,10	120,91
	Memória	1278,81	1279,44	1278,79	1273,27	1276,80	1275,29
Número de pontos	Resultados	*triang_inc*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	37,02	37,38	35,33	35,14	**36,79**	36,54
	Memória	1034,73	1034,82	1034,83	1034,82	1034,83	1034,83
50000	Tempo	38,80	**38,56**	37,15	36,85	39,15	38,48
	Memória	1043,51	1043,54	1043,35	1043,52	1043,60	1043,28
75000	Tempo	40,61	**40,23**	38,35	39,11	40,63	40,87
	Memória	1052,02	1052,12	1052,13	1052,15	1052,50	1051,86
100000	Tempo	42,38	42,38	40,14	40,46	42,83	42,14
	Memória	1061,15	1060,77	1062,01	1061,74	1061,54	1061,74
250000	Tempo	54,07	54,03	51,53	53,05	55,07	54,09
	Memória	1113,64	1113,60	1114,45	1113,66	1113,12	1113,66
500000	Tempo	72,76	77,57	73,13	74,33	77,77	74,65
	Memória	1199,25	1200,47	1201,08	1198,60	1199,62	1197,54
750000	Tempo	92,55	100,08	101,18	102,07	99,33	97,09
	Memória	1285,89	1288,72	1289,01	1282,29	1283,27	1282,54
1000000	Tempo	113,46	124,94	126,89	130,70	122,70	123,02
	Memória	1371,99	1377,42	1380,07	1364,93	1368,60	1367,47

Tabela 4 Resultados de custo computacional (em segundos) e ocupação de memória (em *megabytes*) utilizando a ordem espiral.

Espiral							
Número de pontos	Resultados	*hist_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	**11,92**	3,81	2,30	**7,61**	**8,31**	**5,87**
	Memória	24,13	24,17	24,03	24,17	23,96	23,95
50000	Tempo	27,11	7,13	4,31	**14,87**	**15,70**	**8,73**
	Memória	30,83	30,69	30,42	30,67	30,09	30,33
75000	Tempo	38,90	9,71	**6,31**	24,18	**21,95**	**11,22**
	Memória	36,30	35,65	36,78	35,89	38,19	38,76
100000	Tempo	48,53	11,92	8,88	32,20	**26,60**	**13,75**
	Memória	44,78	44,39	44,93	44,28	44,21	47,46
250000	Tempo	87,29	24,81	**25,02**	74,02	**48,07**	**31,63**
	Memória	81,35	75,23	79,21	78,10	78,27	86,86
500000	Tempo	108,68	54,26	**56,85**	117,16	**83,05**	**63,24**
	Memória	150,33	145,78	132,39	131,36	140,24	148,13
750000	Tempo	118,88	**84,51**	97,96	149,71	**121,10**	**99,05**
	Memória	213,66	212,62	192,77	195,85	197,83	192,35
1000000	Tempo	137,94	130,35	**131,47**	569,96	**157,17**	**136,14**
	Memória	276,58	258,69	239,27	261,55	249,31	261,78
Número de pontos	Resultados	*ult_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	19,62	3,66	**2,20**	17,99	18,77	14,76
	Memória	23,87	23,65	23,52	23,91	23,94	24,19
50000	Tempo	20,36	5,15	**4,18**	22,94	22,75	17,67
	Memória	30,32	29,92	29,33	30,08	30,30	30,28
75000	Tempo	21,61	7,50	6,70	**24,15**	27,00	21,86
	Memória	36,83	35,88	34,78	35,83	36,99	37,03
100000	Tempo	21,71	9,45	**8,83**	**25,20**	30,54	25,36
	Memória	42,75	42,31	40,10	43,30	43,76	42,37
250000	Tempo	28,10	24,74	26,24	**35,48**	55,33	51,63
	Memória	80,57	75,27	69,21	75,41	78,269	78,83
500000	Tempo	49,83	54,74	59,99	**57,32**	102,53	87,71
	Memória	141,68	127,41	112,44	128,58	132,83	131,76
750000	Tempo	74,73	91,19	**97,78**	**86,64**	149,11	123,82
	Memória	201,01	173,98	154,40	182,14	185,97	183,64
1000000	Tempo	99,07	124,64	134,20	**131,56**	201,41	163,37
	Memória	259,11	218,02	199,67	250,86	233,69	232,63
Número de pontos	Resultados	*triang_inc*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	20,17	**3,63**	3,25	18,26	19,27	14,60
	Memória	26,44	26,23	25,85	26,23	26,53	26,52
50000	Tempo	**19,91**	**5,07**	6,88	25,29	24,03	17,51
	Memória	35,20	34,75	34,04	35,05	35,46	35,19
75000	Tempo	**19,86**	**6,81**	11,54	27,13	29,39	23,57
	Memória	44,93	43,87	41,75	43,95	44,69	43,78
100000	Tempo	**18,80**	**8,91**	17,18	32,51	33,37	26,87
	Memória	52,74	51,54	49,39	52,55	53,68	53,84
250000	Tempo	**27,44**	**23,29**	60,89	50,67	71,54	63,04
	Memória	106,71	99,30	92,82	100,30	104,27	102,60
500000	Tempo	**48,10**	**51,16**	168,74	113,22	145,35	119,87
	Memória	193,04	175,84	160,29	177,98	184,45	181,98
750000	Tempo	**70,62**	90,67	322,64	200,68	224,86	192,33
	Memória	276,24	250,11	230,48	255,54	261,90	258,73
1000000	Tempo	**91,33**	**121,74**	508,56	343,33	321,71	284,08
	Memória	360,10	320,50	303,25	330,21	336,00	334,65

Tabela 5 Resultados de custo computacional (em segundos) e ocupação de memória (em *megabytes*) utilizando a ordem da árvore rubro-negra com percurso em ordem. O símbolo - indica que a execução foi abortada após 10 minutos.

Rubro-negra com percurso em ordem							
Número de pontos	Resultados	*hist_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	**4,74**	**2,76**	2,02	**4,06**	**4,33**	**3,39**
	Memória	8,97	9,08	9,08	8,52	9,08	9,08
50000	Tempo	**11,12**	**6,44**	4,30	**9,75**	**9,68**	**7,56**
	Memória	15,83	15,87	16,39	15,69	15,94	16,14
75000	Tempo	**16,55**	**9,91**	7,12	**16,10**	**15,72**	**12,24**
	Memória	22,30	22,99	22,68	21,19	22,32	22,32
100000	Tempo	**26,89**	**11,76**	10,77	**22,62**	**21,65**	**17,19**
	Memória	29,87	29,66	32,51	28,89	30,25	29,92
250000	Tempo	**78,79**	**36,09**	**43,38**	**72,09**	**51,51**	**51,81**
	Memória	70,83	71,85	82,47	67,66	70,18	71,14
500000	Tempo	**189,38**	**105,95**	**152,02**	**176,45**	**87,58**	**120,48**
	Memória	136,01	146,37	188,90	132,05	134,71	138,30
750000	Tempo	**310,02**	**208,89**	**334,41**	**295,08**	**114,86**	**196,04**
	Memória	211,69	238,30	330,64	205,15	208,44	216,09
1000000	Tempo	**472,13**	**354,41**	-	**426,14**	**148,29**	**311,48**
	Memória	270,30	321,45	-	262,80	265,25	276,12
Número de pontos	Resultados	*ult_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	11,79	3,77	2,33	8,12	10,68	6,78
	Memória	7,68	7,79	7,80	7,79	7,79	7,79
50000	Tempo	38,06	8,48	4,95	24,04	29,67	17,21
	Memória	13,51	13,62	13,37	13,62	13,56	13,62
75000	Tempo	83,20	13,86	8,25	45,06	52,99	31,66
	Memória	19,02	19,13	19,13	19,13	19,32	19,13
100000	Tempo	140,32	20,44	11,44	74,77	79,42	49,02
	Memória	24,72	24,82	24,71	24,96	24,96	24,83
250000	Tempo	-	90,98	44,22	396,88	314,78	221,37
	Memória	-	58,89	58,51	58,83	58,83	58,64
500000	Tempo	-	328,91	154,98	-	-	-
	Memória	-	115,51	114,74	-	-	-
750000	Tempo	-	-	338,81	-	-	-
	Memória	-	-	171,09	-	-	-
1000000	Tempo	-	-	**598,38**	-	-	-
	Memória	-	-	227,32	-	-	-
Número de pontos	Resultados	*triang_inc*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	10,97	4,22	**1,83**	5,22	7,33	4,15
	Memória	10,52	10,62	10,37	10,63	10,63	10,63
50000	Tempo	36,59	9,99	**4,11**	16,70	22,88	11,34
	Memória	18,92	18,96	18,71	19,03	19,04	18,78
75000	Tempo	78,50	17,44	**6,99**	33,52	43,74	21,45
	Memória	27,27	27,37	27,57	27,36	27,64	27,38
100000	Tempo	137,19	27,11	**10,48**	57,14	67,53	34,40
	Memória	35,74	36,17	36,69	35,85	35,91	35,66
250000	Tempo	-	138,23	44,65	335,16	298,20	171,41
	Memória	-	92,83	99,44	86,80	86,91	86,26
500000	Tempo	-	522,05	155,05	-	-	-
	Memória	-	203,60	233,97	-	-	-
750000	Tempo	-	-	345,55	-	-	-
	Memória	-	-	418,25	-	-	-
1000000	Tempo	-	-	-	-	-	-
	Memória	-	-	-	-	-	-

Tabela 6 Custo computacional (em segundos) e ocupação de memória (em *megabytes*) utilizando o percurso em largura na árvore rubro-negra. O símbolo - indica que a execução foi abortada após 10 minutos.

Rubro-negra com percurso em largura							
Número de pontos	Resultados	*hist_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	**520,85**	459,34	454,99	**358,77**	**474,92**	**518,52**
	Memória	8,78	8,83	8,84	8,84	8,84	8,84
50000	Tempo	-	-	-	-	-	-
	Memória	-	-	-	-	-	-
75000	Tempo	-	-	-	-	-	-
	Memória	-	-	-	-	-	-
Número de pontos	Resultados	*ult_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	594,60	**164,24**	68,13	462,43	-	-
	Memória	8,83	8,71	9,35	9,10	-	-
50000	Tempo	-	-	286,43	-	-	-
	Memória	-	-	16,08	-	-	-
75000	Tempo	-	-	-	-	-	-
	Memória	-	-	-	-	-	-
Número de pontos	Resultados	*triang_inc*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	-	-	**65,14**	442,46	-	-
	Memória	-	-	10,91	10,65	-	-
50000	Tempo	-	-	**272,88**	-	-	-
	Memória	-	-	16,08	-	-	-
75000	Tempo	-	-	**568,88**	-	-	-
	Memória	-	-	28,30	-	-	-

Tabela 7 Resultados de custo computacional (em segundos) e ocupação de memória (em *megabytes*) sem a ordenação prévia dos pontos. O símbolo - indica que a execução foi abortada após 10 minutos.

Inserção aleatória de pontos							
Número de pontos	Resultados	*hist_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	-	-	-	**410,69**	-	-
	Memória	-	-	-	7,14	-	-
Número de pontos	Resultados	*ult_triang*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	-	-	-	-	-	-
	Memória	-	-	-	-	-	-
Número de pontos	Resultados	*triang_inc*					
		Aleatória	Cruzada	Linha	Cluster	Espiral	Círculo
25000	Tempo	-	-	-	-	-	-
	Memória	-	-	-	-	-	-

4.3 Custos computacionais e ocupações de memória nas ordens de inserção de pontos selecionadas

Nos testes apresentados a seguir, são verificados os custos computacionais e ocupações de memória dos algoritmos incrementais utilizando as abordagens que obtiveram melhor custo computacional nas instâncias de maior tamanho. Esses custos computacionais são comparados com o custo computacional do algoritmo incremental utilizando a ordem da *cut-longest-edge kd-tree*. Em geral, os custos computacionais nas instâncias de tamanho menor coincidem com os custos computacionais nas instâncias de tamanho maior.

Ao utilizar-se a inserção de pontos na ordem da árvore rubro-negra com o percurso em largura e a inserção aleatória de pontos, obteve-se um custo computacional alto para a computação da triangulação de Delaunay se comparado ao custo das demais ordens de inserção de pontos utilizadas neste trabalho. Por esse motivo, essas distribuições não serão utilizadas para a realização dos próximos testes.

Para testes com até 100000 pontos, foram realizadas 5 execuções. Para os demais testes, foram realizadas 3 execuções. Para essas execuções, foi calculada a média do tempo de execução, o desvio padrão e o coeficiente de variação do tempo e a ocupação de memória. Esses resultados, para as 6 distribuições de pontos, são apresentados nas Tabelas de 8 a 13. Note que o menor custo computacional em cada uma das instâncias foi colocado em negrito. Nas Figuras 31, 32 e 33, são apresentados, respectivamente, os resultados de custo computacional e ocupação de memória obtidos nas distribuições aleatória, cruzada e em linha. Nas Figuras 34, 35 e 36,

são apresentados, respectivamente, os resultados de custo computacional e ocupação de memória obtidos nas distribuições em cluster, em espiral e em círculo.

Tabela 8 Custo computacional médio (em segundos) e ocupação de memória (em *megabytes*) dos algoritmos incrementais para a triangulação de Delaunay na distribuição aleatória. Considere n o número de pontos, $\boldsymbol{\sigma}$ o desvio padrão e CV o coeficiente de variação. Máquina utilizada: M1.

n	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert *(ult_triang)*	Curva de Lebesgue *(hist_triang)*	*H-indexing (ult_triang)*	Espiral *(triang_inc)*	Rubro-negra em ordem *(hist_triang)*
25000	Tempo	**1,20**	1,62	2,18	36,62	20,14	4,40
	σ	0,00	0,03	0,07	0,11	0,58	0,22
	CV(%)	0,36	1,92	3,21	0,30	2,89	5,03
	Memória	9,44	24,01	24,90	1032,41	26,44	**8,97**
50000	Tempo	**2,41**	3,27	4,55	38,44	20,21	10,27
	σ	0,03	0,09	0,09	0,14	0,60	0,48
	CV(%)	1,35	2,93	2,05	0,36	2,99	4,76
	Memória	16,72	29,87	31,60	1038,80	35,20	**15,83**
75000	Tempo	**3,63**	4,90	7,12	39,97	19,51	16,48
	σ	0,03	0,08	0,06	0,37	0,49	0,34
	CV(%)	0,93	1,81	0,98	0,94	2,53	2,07
	Memória	23,33	35,89	38,34	1045,13	44,93	**22,30**
100000	Tempo	**4,86**	6,64	9,51	41,72	19,29	24,91
	σ	0,04	0,16	0,18	0,24	0,45	1,11
	CV(%)	0,96	2,48	1,91	0,57	2,37	4,47
	Memória	31,13	41,83	46,26	1051,47	52,74	**29,87**
250000	Tempo	**12,26**	17,10	25,77	52,81	26,86	78,00
	σ	0,27	0,15	0,91	0,83	0,53	0,69
	CV(%)	2,25	0,89	3,54	1,58	1,97	0,88
	Memória	72,43	79,24	86,69	1090,64	106,71	**70,83**
500000	Tempo	**24,70**	34,37	52,97	71,02	47,01	189,49
	σ	0,16	0,67	1,45	0,75	1,11	5,15
	CV(%)	0,65	1,96	2,74	1,05	2,36	2,72
	Memória	142,43	142,17	159,65	1152,98	193,04	**136,01**
750000	Tempo	**37,60**	52,67	80,17	91,03	70,17	324,79
	σ	0,53	1,73	1,06	1,64	1,85	12,98
	CV(%)	1,42	3,29	1,33	1,80	2,64	3,99
	Memória	218,60	**208,23**	233,46	1214,9	276,24	211,69
1000000	Tempo	**50,00**	69,50	109,51	108,92	92,53	477,16
	σ	0,30	0,95	3,49	3,44	2,50	4,41
	CV(%)	0,61	1,37	3,19	3,16	2,70	0,92
	Memória	282,57	276,39	313,41	1278,81	360,10	**270,30**
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	0	1	0	0	0	**7**

Tabela 9 Custo computacional (em segundos) e ocupação de memória (em *megabytes*) dos algoritmos incrementais para a triangulação de Delaunay na distribuição cruzada. Considere n o número de pontos, σ o desvio padrão e CV o coeficiente de variação. Máquina utilizada: M1.

n	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert *(ult_triang)*	Curva de Lebesgue *(hist_triang)*	*H-indexing (ult_triang)*	Espiral *(triang_inc)*	Rubro-negra em ordem *(hist_triang)*
25000	Tempo	**1,21**	1,81	2,15	36,49	3,56	2,70
	σ	0,01	0,03	0,05	0,15	0,11	0,06
	CV(%)	1,40	2,11	2,65	0,43	3,27	2,35
	Memória	9,53	24,28	25,46	1032,50	26,23	**9,08**
50000	Tempo	**2,42**	3,60	4,45	38,41	5,18	6,01
	σ	0,02	0,03	0,07	0,10	0,07	0,46
	CV(%)	1,20	0,84	1,76	0,27	1,43	7,79
	Memória	16,57	30,58	32,51	1038,90	34,75	**15,87**
75000	Tempo	**3,70**	5,55	6,94	40,10	6,87	8,84
	σ	0,04	0,21	0,41	0,33	0,14	0,61
	CV(%)	1,14	3,83	5,97	0,83	2,10	6,92
	Memória	23,46	37,17	39,64	1045,23	43,87	**22,99**
100000	Tempo	**4,93**	7,36	9,53	41,84	8,75	12,08
	σ	0,05	0,17	0,24	0,17	0,17	0,42
	CV(%)	1,16	2,41	2,55	0,40	1,94	3,52
	Memória	31,03	44,63	47,64	1052,34	51,54	**29,66**
250000	Tempo	**12,14**	18,95	26,22	53,00	23,03	35,76
	σ	0,25	0,30	0,67	0,69	0,49	0,85
	CV(%)	2,11	1,59	2,57	1,31	2,15	2,38
	Memória	72,53	88,00	98,37	1089,68	99,30	**71,85**
500000	Tempo	**24,18**	40,67	55,17	72,31	51,76	103,35
	σ	0,32	0,33	1,74	1,38	0,70	2,37
	CV(%)	1,36	0,82	3,17	1,91	1,36	2,29
	Memória	**142,53**	173,53	193,07	1154,02	175,84	146,37
750000	Tempo	**36,17**	67,06	91,25	93,55	87,18	204,62
	σ	0,14	0,99	1,81	2,99	3,03	3,87
	CV(%)	0,40	1,47	1,99	3,19	3,48	1,89
	Memória	**218,72**	268,55	290,16	1216,35	250,11	238,30
1000000	Tempo	**48,18**	95,05	130,41	114,20	122,79	347,01
	σ	0,66	0,55	1,48	6,35	2,17	8,04
	CV(%)	1,38	0,58	1,14	5,56	1,77	2,31
	Memória	**282,80**	369,10	400,87	1279,44	320,50	321,45
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	3	0	0	0	0	**5**

4.4 Análise dos resultados

A abordagem *hist_triang* foi a que apresentou, em geral, melhores custos computacionais, seguida da abordagem *ult_triang*. A abordagem que apresentou piores custos computacionais foi a *triang_inc*.

A inserção de pontos utilizando a ordem do percurso em largura na *cut-longest-edge kd-tree* foi a ordem de inserção que obteve os melhores custos computacionais, em todos os testes. Como o tempo de execução desse algoritmo foi superado pelo algoritmo proposto por Lo (2013), que propôs

Tabela 10 Custo computacional médio (em segundos) e ocupação de memória (em *megabytes*) dos algoritmos incrementais para a triangulação de Delaunay na distribuição em linha. Considere n o número de pontos, σ o desvio padrão e CV o coeficiente de variação. Máquina utilizada: M2.

n	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert *(ult_triang)*	Curva de Lebesgue *(hist_triang)*	*H-indexing (hist_triang)*	Espiral *(hist_triang)*	Rubro-negra em ordem *(ult_triang)*
25000	Tempo	**1,20**	1,62	1,67	34,52	2,28	2,34
	σ	0,00	0,01	0,02	0,40	0,02	0,05
	CV(%)	0,13	0,92	1,59	1,15	1,12	2,39
	Memória	9,55	24,57	25,49	1033,8	24,03	**7,80**
50000	Tempo	**2,43**	3,34	3,63	35,97	4,34	4,95
	σ	0,07	0,05	0,05	0,28	0,13	0,02
	CV(%)	2,99	1,76	1,60	0,79	3,19	0,54
	Memória	16,58	31,61	33,40	1041,04	30,42	**13,37**
75000	Tempo	**3,67**	5,11	5,53	37,63	6,32	7,98
	σ	0,07	0,02	0,03	0,19	0,04	0,15
	CV(%)	1,93	0,55	0,71	0,50	0,66	1,99
	Memória	23,28	39,16	40,32	1047,59	36,78	**19,13**
100000	Tempo	**4,84**	7,05	7,61	39,29	8,58	11,43
	σ	0,05	0,07	0,05	0,23	0,18	0,18
	CV(%)	1,22	1,10	0,78	0,59	2,16	1,65
	Memória	31,23	47,32	50,68	1057,13	44,93	**24,71**
250000	Tempo	**11,88**	19,09	21,31	49,87	24,83	44,31
	σ	0,02	0,07	0,71	0,15	0,40	0,11
	CV(%)	0,19	0,39	3,35	0,31	1,62	0,26
	Memória	72,36	103,79	109,88	1098,89	79,21	**58,51**
500000	Tempo	**23,87**	40,93	47,32	69,61	56,77	153,67
	σ	0,17	0,20	0,80	0,33	0,33	1,21
	CV(%)	0,72	0,50	1,69	0,47	0,59	0,79
	Memória	142,54	213,08	225,12	1171,35	132,39	**114,74**
750000	Tempo	**36,55**	65,66	77,50	92,30	95,32	338,25
	σ	0,46	0,17	1,81	1,15	2,29	1,49
	CV(%)	1,28	0,26	2,33	1,25	2,40	0,44
	Memória	218,41	356,79	372,94	1254,00	192,77	**171,09**
1000000	Tempo	**48,17**	93,88	108,04	116,37	131,86	599,42
	σ	0,34	0,50	0,58	0,72	0,48	1,11
	CV(%)	0,72	0,53	0,53	0,61	0,36	0,18
	Memória	282,28	559,02	580,65	1320,64	239,27	**227,32**
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	0	0	0	0	0	**8**

o esquema *multi-grid*, é provável que o algoritmo de Lo (2013) esteja no estado da arte para a geração da triangulação de Delaunay. A inserção de pontos na ordem da curva de Hilbert apresentou o segundo melhor custo computacional, perdendo para o algoritmo incremental utilizando o percurso em largura na *cut-longest-edge kd-tree*.

Dentre todas as distribuições testadas, a inserção aleatória de pontos foi a ordem de inserção que obteve pior custo computacional, seguida da inserção de pontos na ordem do percurso em largura da árvore rubro-negra. Dentre as 6 ordens de inserção selecionadas para teste, a que apresentou, em

Tabela 11 Custo computacional médio (em segundos) e ocupação de memória (em *megabytes*) dos algoritmos incrementais para a triangulação de Delaunay na distribuição em cluster. Considere n o número de pontos, σ o desvio padrão e CV o coeficiente de variação. Máquina utilizada: M2.

n	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert *(hist_triang)*	Curva de Lebesgue *(hist_triang)*	*H-indexing (ult_triang)*	Espiral *(ult_triang)*	Rubro-negra em ordem *(hist_triang)*
25000	Tempo	**1,22**	1,68	2,43	34,64	17,56	4,11
	σ	0,00	0,05	0,14	0,33	0,24	0,03
	CV(%)	0,50	3,38	5,91	0,97	1,41	0,83
	Memória	9,55	24,85	25,22	1032,50	23,91	**8,52**
50000	Tempo	**2,46**	3,30	4,87	36,33	22,73	9,79
	σ	0,01	0,14	0,08	0,30	0,25	0,11
	CV(%)	0,57	4,46	1,69	0,84	1,13	1,14
	Memória	16,59	32,75	32,41	1038,88	30,08	**15,69**
75000	Tempo	**3,73**	5,01	7,55	38,20	24,21	15,94
	σ	0,03	0,28	0,07	0,31	0,18	0,10
	CV(%)	0,93	5,68	0,99	0,83	0,76	0,63
	Memória	23,27	41,69	39,73	1045,63	35,83	**21,19**
100000	Tempo	**4,98**	6,59	10,28	40,09	25,17	22,62
	σ	0,05	0,07	0,14	0,28	0,10	0,22
	CV(%)	1,01	1,08	1,38	0,70	0,42	0,97
	Memória	31,23	48,78	48,92	1051,62	43,30	**28,89**
250000	Tempo	**12,65**	17,35	27,61	52,14	35,17	71,95
	σ	0,23	0,87	0,59	0,78	0,54	0,30
	CV(%)	1,84	5,02	2,14	1,51	1,55	0,42
	Memória	72,31	96,84	94,20	1089,91	75,41	**67,66**
500000	Tempo	**25,09**	34,32	56,74	73,08	57,90	175,02
	σ	0,18	0,66	0,65	0,35	0,51	1,34
	CV(%)	0,73	1,94	1,14	0,48	0,88	0,76
	Memória	142,57	184,25	180,40	1151,48	**128,58**	132,05
750000	Tempo	**37,54**	52,02	89,22	95,98	87,03	293,74
	σ	0,05	0,77	1,21	0,23	0,89	2,49
	CV(%)	0,14	1,48	1,35	0,24	1,03	0,84
	Memória	218,80	283,28	275,15	1212,10	**182,14**	205,15
1000000	Tempo	**50,59**	70,28	116,90	120,35	130,30	425,94
	σ	0,36	1,67	0,67	0,48	1,09	3,12
	CV(%)	0,72	2,38	0,57	0,39	0,83	0,73
	Memória	282,59	362,35	359,87	1273,27	**250,86**	262,80
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	0	0	0	0	3	**5**

geral, os piores custos computacionais foi a inserção de pontos na ordem do percurso em ordem na árvore rubro-negra. Essa ordem apresentou melhores custos computacionais apenas quando se utilizou a ordem em espiral com os conjuntos de pontos distribuídos em espiral, considerando-se instâncias contendo 1000000 de pontos.

O custo computacional para calcular a ordem do indexador *H-indexing* é alto, independente do tamanho da instância. Além disso, para a computação da ordem dada por esse indexador, é necessário armazenar valores de ordenações calculadas previamente, o que torna o consumo de

Tabela 12 Custo computacional médio (em segundos) e ocupação de memória (em *megabytes*) dos algoritmos incrementais para a triangulação de Delaunay na distribuição espiral. Considere n o número de pontos, σ o desvio padrão e CV o coeficiente de variação. Máquina utilizada: M3.

n	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert *(hist_triang)*	Curva de Lebesgue *(hist_triang)*	*H-indexing (hist_triang)*	Espiral *(hist_triang)*	Rubro-negra em ordem *(hist_triang)*
25000	Tempo	**1,34**	1,76	2,44	36,81	8,31	4,30
	σ	0,00	0,01	0,03	0,28	0,09	0,02
	CV(%)	0,51	0,98	1,60	0,77	1,10	0,67
	Memória	9,54	24,71	24,98	1033,54	23,96	**9,08**
50000	Tempo	**2,72**	3,56	5,14	38,57	15,70	9,72
	σ	0,02	0,03	0,22	0,24	0,26	0,06
	CV(%)	0,90	1,07	4,39	0,63	1,66	0,65
	Memória	16,75	30,66	31,86	1041,02	30,09	**15,94**
75000	Tempo	**4,12**	5,78	7,95	40,41	21,40	15,43
	σ	0,07	0,23	0,48	0,14	0,33	0,16
	CV(%)	1,72	4,11	6,12	0,34	1,55	1,08
	Memória	23,51	39,26	39,13	1047,54	38,19	**22,32**
100000	Tempo	**5,46**	7,24	10,52	42,31	26,12	21,19
	σ	0,01	0,06	0,22	0,13	0,32	0,34
	CV(%)	0,34	0,96	2,08	0,31	1,22	1,64
	Memória	31,17	44,15	46,10	1056,30	44,21	**30,25**
250000	Tempo	**13,40**	18,55	27,51	53,94	48,53	52,01
	σ	0,17	0,27	0,21	0,56	0,43	0,51
	CV(%)	1,32	1,48	0,77	1,04	0,89	0,98
	Memória	72,44	89,00	91,94	1099,63	78,27	**70,18**
500000	Tempo	**26,63**	37,05	57,71	73,78	83,22	87,23
	σ	0,16	0,39	0,48	0,08	0,16	0,34
	CV(%)	0,60	1,06	0,83	0,11	0,20	0,39
	Memória	142,58	169,92	176,24	1168,93	140,24	**134,71**
750000	Tempo	**40,40**	55,77	87,54	96,11	120,51	115,99
	σ	0,27	1,01	1,27	2,21	0,60	1,30
	CV(%)	0,68	1,82	1,45	2,30	0,49	1,12
	Memória	218,75	253,83	277,37	1248,5	**197,83**	208,44
1000000	Tempo	**53,59**	73,83	119,74	116,67	158,77	147,72
	σ	0,35	0,16	3,14	0,93	1,44	0,70
	CV(%)	0,65	0,22	2,62	0,80	0,91	0,47
	Memória	282,77	351,61	364,80	1312,69	**249,31**	265,25
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	0	0	0	0	2	**6**

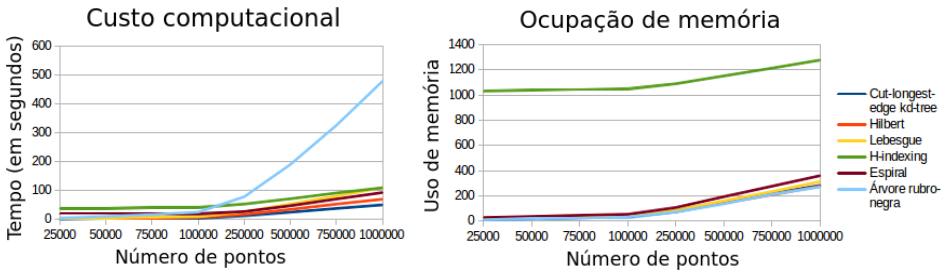
memória alto ao se utilizar essa ordem de inserção de pontos.

Ao se utilizar a ordem espiral, pode ocorrer que instâncias de tamanho maior sejam executadas mais rapidamente que instâncias de tamanho menor. Isso ocorre nessa ordem pois os pontos tendem a ficar mais próximos entre si quando há uma quantidade maior de pontos, o que diminui o tempo para a busca de um circuncírculo que contém o novo ponto. Por esse motivo observa-se, por exemplo, que instâncias com 75000 pontos são executadas mais rapidamente que instâncias com 25000 pontos, na distribuição aleatória de pontos (tabela 8).

Tabela 13 Média do custo computacional (em segundos) e ocupação de memória (em *megabytes*) dos 6 algoritmos incrementais para a triangulação de Delaunay na distribuição em círculo. Considere n o número de pontos, σ o desvio padrão e CV o coeficiente de variação. Máquina utilizada para teste: M3.

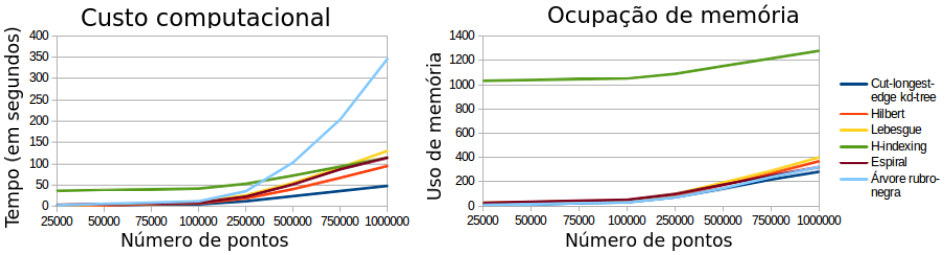
n	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert *(ult_triang)*	Curva de Lebesgue *(hist_triang)*	*H-indexing (hist_triang)*	Espiral *(hist_triang)*	Rubro-negra em ordem *(hist_triang)*
25000	Tempo	**1,30**	1,72	2,13	36,62	5,80	3,40
	σ	0,00	0,03	0,03	0,15	0,05	0,05
	CV(%)	0,26	1,91	1,43	0,42	0,90	1,65
	Memória	9,55	24,34	25,23	1033,54	23,95	**9,08**
50000	Tempo	**2,63**	3,50	4,58	38,34	8,86	7,59
	σ	0,02	0,04	0,07	0,09	0,10	0,05
	CV(%)	0,95	1,34	1,73	0,23	1,17	0,75
	Memória	16,58	30,35	31,82	1040,96	30,33	**16,14**
75000	Tempo	**3,90**	5,26	7,08	40,09	11,38	12,40
	σ	0,02	0,41	0,10	0,19	0,10	0,13
	CV(%)	0,67	7,96	1,55	0,48	0,91	1,11
	Memória	23,48	37,33	38,99	1047,28	38,76	**22,32**
100000	Tempo	**5,25**	7,13	9,61	42,00	13,88	17,28
	σ	0,06	0,15	0,09	0,20	0,12	0,13
	CV(%)	1,21	2,16	0,97	0,49	0,88	0,75
	Memória	31,30	43,53	46,97	1056,29	47,46	**29,92**
250000	Tempo	**13,11**	18,36	26,30	53,35	31,76	51,90
	σ	0,04	0,48	0,52	0,32	0,16	0,10
	CV(%)	0,31	2,61	1,98	0,61	0,52	0,20
	Memória	72,28	84,93	94,38	1099,75	86,86	**71,14**
500000	Tempo	**26,73**	36,38	53,61	73,52	63,90	120,12
	σ	0,04	0,76	1,20	0,30	0,80	0,38
	CV(%)	0,15	2,09	2,24	0,42	1,26	0,32
	Memória	142,60	161,64	176,93	1170,36	148,13	**138,30**
750000	Tempo	**40,02**	55,21	83,69	95,18	98,08	197,59
	σ	0,47	0,98	1,37	0,75	0,87	2,28
	CV(%)	1,17	1,78	1,63	0,79	0,89	1,15
	Memória	218,91	247,14	272,16	1248,77	**192,35**	216,09
1000000	Tempo	**53,36**	74,65	112,76	117,84	134,47	292,10
	σ	0,04	0,43	0,68	1,17	1,76	16,78
	CV(%)	0,08	0,57	0,61	0,99	1,31	5,74
	Memória	282,70	336,52	366,31	1310,18	**261,78**	276,12
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	0	0	0	0	2	**6**

A curva de Lebesgue apresentou melhores resultados ao utilizar a abordagem *hist_triang*. A abordagem *ult_triang* utilizou menor quantidade de memória. Em geral, a inserção de pontos pela ordem da árvore rubro-negra com percurso em ordem apresentou as menores ocupações de memória. Porém, a ocupação de memória do algoritmo com a inserção de pontos na ordem da *cut-longest-edge kd-tree* foi competitiva com a ocupação de memória das demais ordens testadas. É possível que em uma outra implementação, o algoritmo utilizando a ordem da *cut-longest-edge kd-tree* utilize uma ocupação de memória menor que a utilizada neste trabalho.



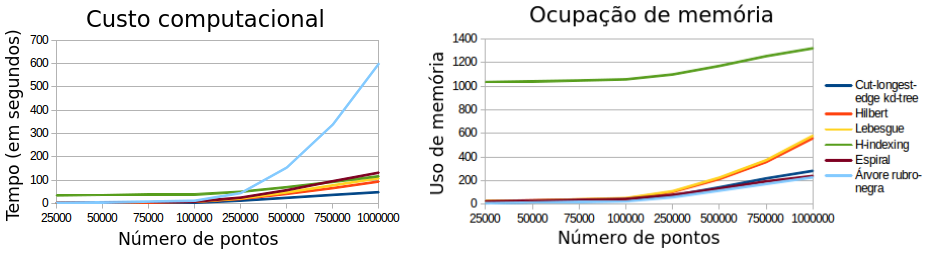
(a) Média de custo computacional. (b) Ocupação de memória.

Figura 31 Média de custo computacional e ocupação de memória dos 6 algoritmos para a malha de Delaunay na distribuição aleatória.



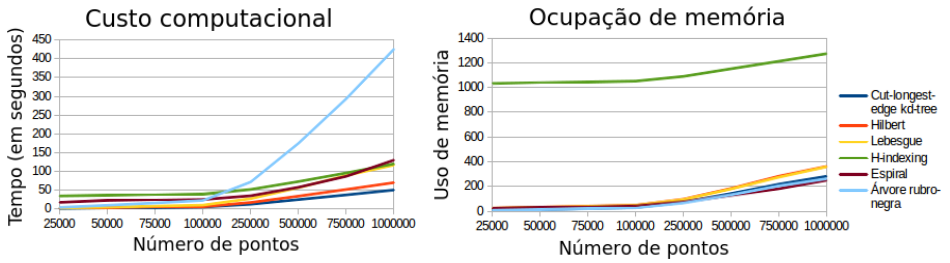
(a) Média de custo computacional. (b) Ocupação de memória.

Figura 32 Média de custo computacional e ocupação de memória dos 6 algoritmos incrementais para a triangulação de Delaunay na distribuição cruzada.



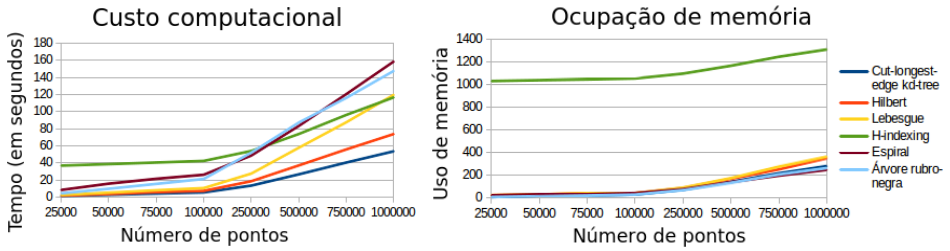
(a) Média de custo computacional. (b) Ocupação de memória.

Figura 33 Média de custo computacional e ocupação de memória dos 6 algoritmos incrementais para a triangulação de Delaunay na distribuição em linha.



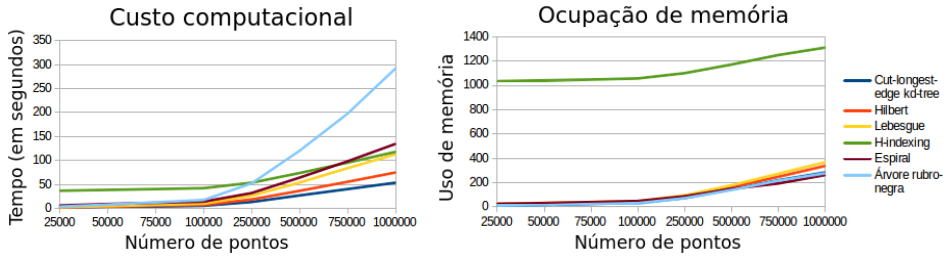
(a) Média de custo computacional. (b) Ocupação de memória.

Figura 34 Média de custo computacional e ocupação de memória dos algoritmos incrementais para a malha de Delaunay na distribuição em cluster.



(a) Média de custo computacional. (b) Ocupação de memória.

Figura 35 Custo computacional e ocupação de memória dos algoritmos incrementais para a malha de Delaunay na distribuição em espiral.



(a) Média de custo computacional. (b) Ocupação de memória.

Figura 36 Custo computacional e ocupação de memória dos algoritmos incrementais para a malha de Delaunay na distribuição em círculo.

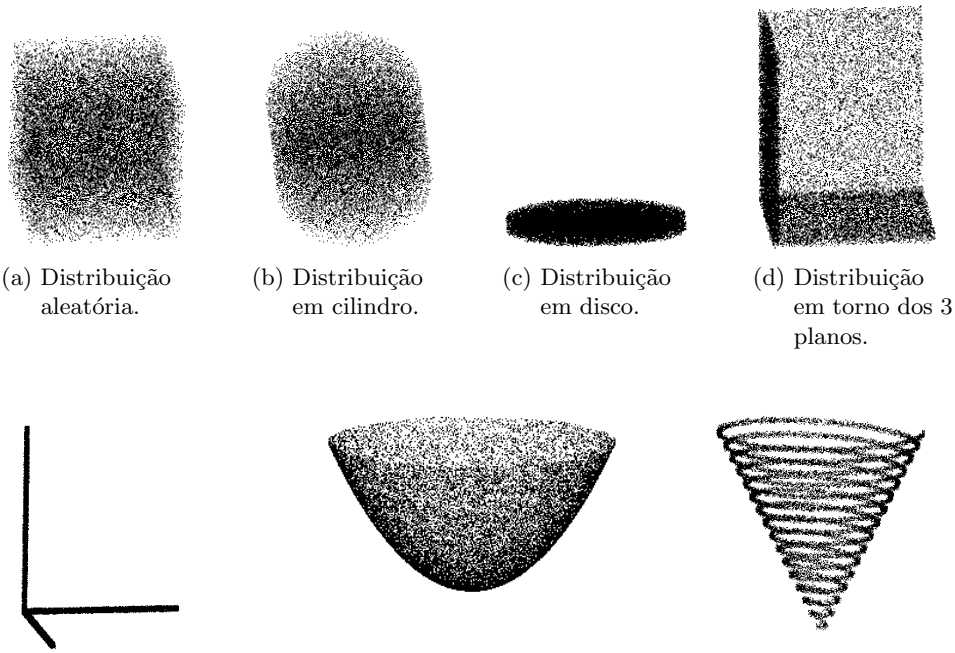
5 RESULTADOS PARA A TETRAEDRIZAÇÃO DE DELAUNAY E ANÁLISES

Nesta seção, são apresentadas as distribuições de pontos utilizadas em testes e os resultados de custo computacional e utilização de memória obtidos nas 8 sequências de inserção de pontos nesses conjuntos de pontos. Na subseção 5.1, são apresentadas as 7 distribuições de pontos utilizadas em testes. Na subseção 5.2, são apresentados os custos computacionais e ocupações de memória nas diferentes ordens de inserção de pontos, exceto pela ordem dada pela *cut-longest-edge kd-tree*, utilizando três diferentes abordagens para a inserção de pontos. Na subseção 5.3, são apresentadas as médias de custo computacional, ocupação de memória, desvio padrão e coeficiente de variação nas ordens de inserção de pontos selecionadas para teste. Finalmente, na seção 5.4, conclusões sobre os testes realizados são apresentadas.

5.1 Distribuições de pontos utilizadas para testes

Nos testes tridimensionais, foram utilizadas 7 distribuições de pontos: aleatória, em cilindro, em disco, nos planos, nos eixos, parabolóide e espiral. Essas distribuições são apresentadas na Figura 37.

Os conjuntos de pontos utilizados contêm entre 25000 e 1000000 de pontos e as coordenadas dos pontos estão no intervalo unitário. Uma vez que os testes ocorreram no intervalo unitário, a biblioteca de precisão GNU MPFR com 256 *bits* de precisão foi utilizada para tornar possível obter uma maior precisão nos testes de posicionamento e da circunsfera.



(a) Distribuição aleatória.

(b) Distribuição em cilindro.

(c) Distribuição em disco.

(d) Distribuição em torno dos 3 planos.

(e) Distribuição em torno dos 3 eixos.

(f) Distribuição em parabolóide.

(g) Distribuição em espiral.

Figura 37 Distribuições de pontos utilizadas em testes. Esses conjuntos contêm 25000 pontos.

5.2 Resultados empíricos utilizando 3 formas para a busca de pontos

Assim como nos algoritmos bidimensionais, nos algoritmos tridimensionais, exceto para o algoritmo utilizando a inserção de pontos pela ordem dada pela *cut-longest-edge kd-tree*, 3 abordagens para a busca de um ponto p inserido na tesselação foram testadas. Essas 3 abordagens são:

- *hist_tet*: tetraedros são verificados na ordem inversa em que eles foram

inseridos na tesselação.

- *ult_tet*: realiza-se uma busca em largura que é iniciada pelo último tetraedro criado.

- *tet_inc*: k tetraedros incidentes ao último ponto p inserido na tesselação são verificados. Se um tetraedro cuja circunsfera contendo o ponto p não for encontrado, tetraedros adjacentes aos k tetraedros também são verificados, a partir do tetraedro menos recentemente verificado.

A busca pelo ponto p termina quando se encontra um tetraedro cuja circunsfera contenha p. Para o algoritmo incremental utilizando a inserção de pontos na ordem dada pela *cut-longest-edge kd-tree*, a busca pela circunsfera que passa sobre um tetraedro e que contém p é realizada pelos k tetraedros incidentes ao nó pai de p na *cut-longest-edge kd-tree*.

Os resultados obtidos são apresentados a seguir. Nas Tabelas entre 14 e 20 são apresentados os resultados de uma execução para as 7 diferentes ordens de inserção, nas 3 diferentes abordagens. Note que o menor custo computacional em cada uma das instâncias foi colocado em negrito. Nessas tabelas, considere que o símbolo "-" indica que a execução foi abortada. Em estruturas tridimensionais, uma execução foi abortada após 60 minutos de execução do algoritmo.

Tabela 14 Custo computacional (em segundos) e ocupação de memória (em *megabytes*) do algoritmo incremental utilizando a ordem da curva de Hilbert, com as 3 abordagens para a busca do tetraedro que contém o novo ponto inserido, nas 7 distribuições de pontos.

Inserção na ordem da curva de Hilbert								
Número de pontos	Resultados	*hist_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	22,14	21,10	23,09	**20,67**	21,29	23,15	21,68
	Memória	38,50	37,55	36,67	36,78	44,81	43,04	36,60
50000	Tempo	45,34	44,06	47,24	43,36	44,46	47,37	42,30
	Memória	62,93	64,28	64,70	62,78	75,51	64,80	66,55
75000	Tempo	70,11	67,33	72,45	70,86	68,17	75,84	67,62
	Memória	88,96	93,15	93,39	88,51	104,32	92,80	94,20
100000	Tempo	92,20	95,18	97,62	95,04	91,28	103,57	87,05
	Memória	114,15	113,32	118,56	115,48	135,59	120,19	122,02
250000	Tempo	236,67	231,88	281,52	225,99	238,12	269,88	230,41
	Memória	268,11	268,02	282,90	276,60	302,46	295,44	293,17
500000	Tempo	482,69	488,00	529,39	476,33	511,13	629,51	496,28
	Memória	516,43	522,32	580,12	556,84	567,85	568,66	576,80
750000	Tempo	757,86	883,23	856,03	777,26	772,49	1065,41	781,06
	Memória	787,95	781,76	889,77	852,22	836,67	840,98	859,72
1000000	Tempo	982,23	1061,26	1206,12	1081,69	1205,82	1587,46	1157,90
	Memória	1046,48	1050,52	1174,92	1139,43	1084,06	1098,69	1119,75
Número de pontos	Resultados	*ult_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	24,33	23,82	25,32	24,20	23,58	27,77	25,43
	Memória	33,59	33,94	33,94	35,92	37,72	36,80	36,57
50000	Tempo	50,59	49,04	52,61	48,75	49,57	56,74	48,47
	Memória	56,96	57,04	55,03	55,87	64,73	59,26	61,24
75000	Tempo	76,58	75,01	80,39	74,01	76,07	86,21	72,31
	Memória	75,37	76,03	79,19	76,92	91,13	86,09	86,26
100000	Tempo	104,09	100,59	109,76	98,56	102,73	115,12	98,21
	Memória	96,83	98,21	101,57	97,54	119,26	108,92	111,27
250000	Tempo	267,03	261,46	280,76	257,12	267,57	295,36	250,54
	Memória	224,81	227,68	247,44	239,49	259,55	276,41	267,91
500000	Tempo	544,19	537,15	563,49	523,38	579,46	611,63	514,47
	Memória	440,20	444,97	508,07	489,86	490,05	537,66	534,35
750000	Tempo	819,24	798,95	868,96	787,85	888,58	964,40	780,91
	Memória	663,01	667,81	775,00	744,29	709,88	785,71	774,62
1000000	Tempo	1100,95	1077,82	1186,30	1058,02	1336,34	1264,37	1142,13
	Memória	888,45	897,89	1040,79	1005,25	932,92	1039,33	1026,06
Número de pontos	Resultados	*tet_inc*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	**21,31**	**20,48**	**22,03**	20,68	**20,44**	**22,55**	**20,13**
	Memória	32,83	33,43	33,42	32,76	37,21	34,37	34,11
50000	Tempo	**43,89**	**42,82**	**47,20**	**41,81**	**42,34**	**47,05**	**41,88**
	Memória	55,43	54,07	55,55	54,65	66,45	59,52	58,47
75000	Tempo	**67,61**	**64,11**	**69,49**	**64,92**	**65,96**	**71,26**	**62,73**
	Memória	74,53	75,50	78,17	78,20	91,39	86,56	86,79
100000	Tempo	**88,82**	**85,74**	**93,18**	**85,62**	**87,32**	**95,08**	**82,66**
	Memória	98,78	95,20	99,25	96,94	118,03	111,69	110,25
250000	Tempo	**227,88**	**221,18**	**240,12**	**219,64**	**227,68**	**244,83**	**215,87**
	Memória	222,75	222,14	249,33	237,69	260,52	280,50	267,91
500000	Tempo	**464,95**	**462,81**	**489,48**	**442,11**	**490,93**	**509,93**	**438,99**
	Memória	438,82	446,07	503,02	487,96	490,46	537,74	532,32
750000	Tempo	**696,00**	**683,51**	**733,58**	**679,22**	**756,31**	**825,51**	**656,16**
	Memória	660,46	666,99	768,66	741,77	710,39	788,91	776,41
1000000	Tempo	**929,73**	**925,99**	**1008,19**	**894,22**	**1174,31**	**1060,65**	**967,16**
	Memória	887,34	895,78	1038,47	1007,83	931,39	1038,68	1026,84

5.3 Custos computacionais e ocupações de memória nas ordens de inserção de pontos selecionadas

Nesses novos testes, são verificados os custos computacionais e ocupações de memória dos algoritmos incrementais utilizando as abordagens que obtiveram melhor custo computacional nas instâncias de maior

Tabela 15 Custo computacional (em segundos) e ocupação de memória (em *megabytes*) do algoritmo incremental utilizando a ordem da curva de Lebesgue, com as 3 abordagens para a busca do tetraedro que contém o novo ponto inserido, nas 7 distribuições de pontos.

Inserção na ordem da curva de Lebesgue								
Número de pontos	Resultados	*hist_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Paraboloide	Espiral
25000	Tempo	**29,48**	**28,28**	**29,55**	**26,16**	25,53	**29,56**	**24,80**
	Memória	40,52	37,15	35,98	35,25	40,24	37,84	37,75
50000	Tempo	**61,32**	**58,63**	**62,16**	**54,08**	55,33	**61,74**	**53,82**
	Memória	57,84	56,88	59,33	58,94	68,87	63,57	63,35
75000	Tempo	**93,29**	**90,93**	**96,63**	**84,16**	86,24	**95,04**	**79,12**
	Memória	83,44	83,91	86,45	83,85	85,08	95,15	91,41
100000	Tempo	**126,64**	**127,86**	**131,25**	**115,24**	117,94	**127,16**	**107,38**
	Memória	113,33	105,38	111,33	106,58	125,05	122,16	117,81
250000	Tempo	**344,20**	**328,04**	**354,35**	**313,50**	**318,49**	**340,21**	**294,34**
	Memória	255,76	242,97	263,92	252,56	278,13	286,61	283,44
500000	Tempo	**709,51**	**697,08**	**732,17**	**644,63**	**714,07**	**713,79**	**614,10**
	Memória	488,63	474,74	538,93	517,14	523,22	568,26	560,37
750000	Tempo	**1093,10**	**1049,40**	**1164,80**	**988,34**	**1109,81**	**1189,07**	**955,24**
	Memória	720,46	718,96	823,42	796,64	765,28	824,62	830,92
1000000	Tempo	**1487,05**	**1508,01**	**1577,89**	**1347,87**	**1661,08**	**1581,63**	**1368,22**
	Memória	943,60	960,59	1100,35	1059,63	992,28	1090,79	1075,11
Número de pontos	Resultados	*ult_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Paraboloide	Espiral
25000	Tempo	40,95	39,32	39,04	32,84	29,63	37,34	31,00
	Memória	39,00	36,63	36,63	34,41	37,87	35,64	37,83
50000	Tempo	97,99	97,25	88,82	73,04	65,60	79,19	66,82
	Memória	57,06	57,65	59,22	57,84	65,19	61,64	60,57
75000	Tempo	165,08	152,22	141,88	118,34	109,73	122,15	102,64
	Memória	82,75	78,25	83,79	84,55	93,39	86,17	87,27
100000	Tempo	228,31	225,51	207,63	160,26	151,87	168,57	137,70
	Memória	107,15	103,43	106,33	100,89	119,13	113,72	111,82
250000	Tempo	737,54	709,74	625,30	482,01	453,52	474,94	390,32
	Memória	248,66	241,46	259,18	248,08	263,94	285,92	273,90
500000	Tempo	1776,50	1729,19	1534,35	1112,70	1109,93	1035,26	853,18
	Memória	484,12	486,65	520,33	496,20	500,18	551,77	556,21
750000	Tempo	3010,64	2943,30	2683,74	1818,62	1875,68	1669,23	1380,80
	Memória	718,12	717,62	801,62	757,39	728,87	822,37	804,46
1000000	Tempo	-	-	-	2506,53	2913,19	2329,67	2015,86
	Memória	-	-	-	1027,47	947,41	1073,43	1043,65
Número de pontos	Resultados	*tet_inc*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Paraboloide	Espiral
25000	Tempo	36,11	34,18	33,75	27,28	**23,41**	30,90	26,10
	Memória	36,28	35,67	35,86	35,00	39,19	35,06	36,87
50000	Tempo	85,64	85,38	74,02	58,93	**51,16**	66,61	55,71
	Memória	59,52	59,59	56,57	54,87	65,20	61,26	61,54
75000	Tempo	142,09	136,45	120,50	96,37	**84,03**	102,30	85,47
	Memória	82,97	78,32	78,40	78,82	91,68	87,88	88,25
100000	Tempo	206,98	204,39	173,11	141,08	**114,43**	141,94	113,29
	Memória	104,85	105,10	104,57	101,94	116,61	115,18	112,73
250000	Tempo	669,64	659,72	527,03	403,77	360,04	399,67	333,58
	Memória	248,55	244,88	256,88	248,57	259,13	285,90	270,55
500000	Tempo	1679,45	1582,24	1302,47	1121,49	880,34	867,75	712,80
	Memória	489,42	488,40	521,63	502,94	493,43	554,07	558,03
750000	Tempo	2825,06	2705,95	2282,08	1534,19	1515,28	1406,49	1162,60
	Memória	725,75	727,16	791,57	768,40	719,29	823,23	809,23
1000000	Tempo	-	-	3376,47	2133,91	2383,29	1978,72	1682,98
	Memória	-	-	1068,08	1031,91	952,30	1077,52	1045,77

tamanho. Esses custos computacionais são comparados com o custo computacional do algoritmo incremental utilizando a inserção de pontos na ordem da *cut-longest-edge kd-tree*. Em geral, os custos computacionais nas instâncias de tamanho menor coincidem com os custos computacionais nas instâncias de tamanho maior.

Assim como em estruturas bidimensionais, em estruturas

Tabela 16 Custo computacional (em segundos) e ocupação de memória (em *megabytes*) do algoritmo incremental utilizando a ordem da *H-indexing*, com as 3 abordagens para a busca do tetraedro que contém o novo ponto inserido, nas 7 distribuições de pontos.

Inserção na ordem *H-indexing*								
Número de pontos	Resultados	*hist_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	51,30	49,69	72,57	51,39	59,71	42,74	40,95
	Memória	32,09	33,51	35,07	36,30	35,44	31,62	32,18
50000	Tempo	112,93	110,08	161,34	115,06	132,31	94,15	91,83
	Memória	64,33	62,87	64,82	65,25	62,89	59,08	62,25
75000	Tempo	180,36	169,33	259,72	184,21	210,46	147,69	139,97
	Memória	93,10	93,27	96,43	92,88	95,72	91,05	88,87
100000	Tempo	249,63	238,26	363,35	257,35	296,80	203,69	193,98
	Memória	120,15	122,96	129,83	127,69	123,82	116,91	117,97
250000	Tempo	713,13	660,85	1108,54	754,32	**846,28**	553,67	532,16
	Memória	302,12	302,56	305,11	298,66	302,55	290,96	289,78
500000	Tempo	1607,39	1442,18	**2547,78**	1728,25	1822,91	1191,40	1121,61
	Memória	599,47	598,74	613,43	597,52	601,11	603,43	586,72
750000	Tempo	2607,18	2440,62	-	**2724,49**	2802,25	1842,12	1772,78
	Memória	914,15	909,96	-	913,16	905,78	895,97	896,29
1000000	Tempo	-	3466,20	-	-	-	2525,29	2399,38
	Memória	-	1207,96	-	-	-	1215,43	1172,43
Número de pontos	Resultados	*ult_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	114,56	111,07	224,62	120,20	167,69	84,44	75,99
	Memória	26,12	25,57	26,57	25,66	29,08	25,52	28,73
50000	Tempo	288,31	260,10	599,46	315,83	453,58	203,98	186,88
	Memória	47,75	50,24	48,46	51,04	50,85	45,36	47,76
75000	Tempo	488,42	445,39	1073,60	547,81	821,11	337,20	307,61
	Memória	71,49	70,76	71,75	70,10	73,18	73,88	73,95
100000	Tempo	727,99	648,83	1646,92	814,47	1241,80	490,13	452,47
	Memória	90,95	90,10	94,32	96,15	96,39	89,75	92,60
250000	Tempo	2539,56	2285,81	-	3037,38	-	1366,21	1289,00
	Memória	222,02	222,35	-	223,88	-	233,22	229,37
500000	Tempo	-	-	-	-	-	2829,21	2686,31
	Memória	-	-	-	-	-	462,75	439,55
750000	Tempo	-	-	-	-	-	-	-
	Memória	-	-	-	-	-	-	-
1000000	Tempo	-	-	-	-	-	-	-
	Memória	-	-	-	-	-	-	-
Número de pontos	Resultados	*tet_inc*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	**41,54**	**40,78**	**58,18**	**42,69**	**50,91**	**34,64**	**32,83**
	Memória	27,15	24,34	27,02	28,88	25,34	28,75	27,18
50000	Tempo	**91,20**	**89,30**	**132,75**	**96,50**	**116,27**	**75,52**	**72,82**
	Memória	48,62	51,15	47,22	49,79	50,33	46,23	46,46
75000	Tempo	**143,44**	**141,56**	**220,46**	**155,66**	**196,26**	**118,73**	**117,38**
	Memória	69,69	70,80	67,05	67,04	72,38	74,00	70,05
100000	Tempo	**199,75**	**190,92**	**323,48**	**220,02**	**288,85**	**164,67**	**161,38**
	Memória	96,23	89,42	91,76	95,21	98,45	89,53	92,41
250000	Tempo	**577,81**	**558,37**	**1095,41**	**687,12**	875,84	**465,56**	**448,41**
	Memória	220,36	216,48	220,36	221,83	224,51	231,02	230,15
500000	Tempo	**1364,31**	**1272,59**	2632,19	**1699,99**	**1531,34**	**1007,96**	**982,32**
	Memória	433,87	431,97	443,44	431,04	447,18	462,44	436,21
750000	Tempo	**2281,44**	**2131,05**	-	2778,73	**2106,72**	**1610,85**	**1514,24**
	Memória	644,29	642,23	-	659,05	655,30	679,82	670,73
1000000	Tempo	**3294,13**	**3037,27**	-	-	**2737,21**	**2230,89**	**2142,49**
	Memória	865,55	861,63	-	-	876,79	924,184	898,90

tridimensionais, ao utilizar-se a inserção de pontos na ordem da árvore rubro-negra com o percurso em largura e a inserção aleatória de pontos, obteve-se um custo computacional alto para a computação da malha de Delaunay. Para algumas instâncias, esses algoritmos foram executados em conjuntos com até 25000 pontos. Nas demais instâncias, os testes excederam 60 minutos.

Tabela 17 Custo computacional (em segundos) e ocupação de memória (em *megabytes*) do algoritmo incremental utilizando a ordem espiral, com as 3 abordagens para a busca do tetraedro que contém o novo ponto inserido, nas 7 distribuições de pontos.

Inserção na ordem espiral								
Número de pontos	Resultados	*hist_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Paraboloíde	Espiral
25000	Tempo	50,70	48,04	70,67	51,45	59,35	42,93	40,48
	Memória	33,28	33,26	36,14	33,48	35,42	34,90	31,43
50000	Tempo	112,83	106,21	159,18	116,12	130,13	93,85	90,61
	Memória	63,74	60,76	69,15	63,40	64,48	57,78	60,34
75000	Tempo	179,16	172,59	255,83	183,34	212,19	148,60	141,12
	Memória	94,23	91,75	96,85	94,17	94,94	93,76	89,38
100000	Tempo	248,97	241,67	360,51	261,11	**295,04**	206,92	195,36
	Memória	119,53	122,26	124,04	122,02	125,37	116,10	119,42
250000	Tempo	704,14	685,43	**1103,84**	752,53	**878,21**	591,39	560,41
	Memória	300,02	298,66	308,06	301,78	304,55	284,67	289,03
500000	Tempo	1586,39	1521,66	**2607,23**	**1718,96**	**2014,75**	1350,36	1284,64
	Memória	600,28	602,68	607,53	599,75	601,17	574,59	577,99
750000	Tempo	2601,83	2496,68	-	**2792,24**	**3302,04**	2212,26	2056,10
	Memória	915,43	914,60	-	910,23	925,30	876,56	886,57
1000000	Tempo	-	3488,91	-	-	-	**3104,44**	**2860,85**
	Memória	-	1198,61	-	-	-	1148,20	1156,00
Número de pontos	Resultados	*ult_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Paraboloíde	Espiral
25000	Tempo	113,76	104,86	225,84	126,90	170,01	83,98	76,17
	Memória	24,88	25,65	29,99	26,90	25,39	24,87	26,08
50000	Tempo	284,92	263,09	591,19	317,58	459,72	202,42	188,76
	Memória	52,53	46,98	47,73	47,80	50,91	49,85	46,19
75000	Tempo	494,86	450,91	1075,68	545,39	860,04	350,47	322,59
	Memória	68,71	66,55	71,45	70,39	73,37	74,03	69,89
100000	Tempo	725,62	685,11	763,40	816,87	1333,91	489,58	474,76
	Memória	93,95	94,41	90,92	95,30	98,24	91,88	88,63
250000	Tempo	2541,91	2246,73	-	3138,53	-	1658,91	1596,43
	Memória	219,27	218,64	-	221,66	-	233,02	229,70
500000	Tempo	-	-	-	-	-	-	-
	Memória	-	-	-	-	-	-	-
750000	Tempo	-	-	-	-	-	-	-
	Memória	-	-	-	-	-	-	-
1000000	Tempo	-	-	-	-	-	-	-
	Memória	-	-	-	-	-	-	-
Número de pontos	Resultados	*tet_inc*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Paraboloíde	Espiral
25000	Tempo	**41,47**	**40,24**	**57,53**	**43,33**	**50,46**	**34,27**	**33,74**
	Memória	24,98	27,58	25,14	25,65	25,91	26,30	25,57
50000	Tempo	**90,94**	**86,63**	**134,37**	**95,60**	**122,20**	**75,46**	**74,09**
	Memória	49,40	49,75	46,37	48,83	49,55	48,06	47,68
75000	Tempo	**142,76**	**138,95**	**223,03**	**153,88**	**203,44**	**119,34**	**115,34**
	Memória	67,44	71,49	71,48	68,91	69,30	69,53	69,35
100000	Tempo	**199,02**	**191,57**	**320,28**	**224,38**	302,49	**164,42**	**162,00**
	Memória	89,30	87,83	89,81	92,15	92,76	87,33	90,57
250000	Tempo	**574,69**	**560,11**	1159,02	**738,30**	1188,90	**478,09**	**473,01**
	Memória	214,27	217,93	217,96	223,68	226,02	232,73	224,42
500000	Tempo	**1345,04**	**1281,15**	3580,11	2899,57	-	**1195,49**	**1159,24**
	Memória	429,25	441,11	425,87	434,21	-	460,49	438,02
750000	Tempo	**2280,03**	**2117,48**	-	-	-	**2088,29**	**2040,91**
	Memória	644,047	642,04	-	-	-	679,18	671,26
1000000	Tempo	**3346,59**	**3104,88**	-	-	-	3217,54	3200,86
	Memória	854,19	857,60	-	-	-	873,38	899,44

Para testes com até 100000 pontos, foram realizadas cinco execuções. Para testes com mais de 100000 pontos, foram realizadas três execuções. Para essas execuções, foi calculada a média do tempo de execução, que é apresentada em segundos, o desvio padrão e o coeficiente de variação do tempo e a ocupação de memória, que é dada em *megabytes*.

Tabela 18 Custo computacional (em segundos) e ocupação de memória (em *megabytes*) do algoritmo incremental utilizando o percurso em ordem na árvore rubro-negra, nas 3 abordagens para a busca do tetraedro que contém o novo ponto inserido, nas 7 distribuições de pontos.

Inserção na ordem da árvore rubro-negra com percurso em ordem								
Número de pontos	Resultados	*hist_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	51,54	48,64	46,29	50,98	60,50	40,12	37,74
	Memória	30,65	35,16	30,78	33,34	37,01	32,60	30,26
50000	Tempo	113,39	109,78	102,16	115,97	138,84	88,10	82,37
	Memória	60,17	60,69	59,62	60,46	61,57	55,10	57,86
75000	Tempo	181,64	171,94	164,19	183,63	**221,68**	137,68	133,03
	Memória	89,22	89,16	86,63	90,41	89,37	82,41	83,73
100000	Tempo	253,46	238,67	227,42	260,14	**316,29**	190,55	182,06
	Memória	115,44	115,18	116,57	115,23	115,84	109,30	111,37
250000	Tempo	734,63	703,64	663,62	**797,19**	**933,54**	531,62	523,44
	Memória	285,35	282,94	280,25	286,41	285,31	268,71	273,44
500000	Tempo	1683,80	1592,40	1475,10	**1796,26**	**2154,09**	1112,73	1101,83
	Memória	565,89	570,34	556,55	566,37	568,66	540,71	544,56
750000	Tempo	2755,78	**2611,90**	2539,55	**2976,46**	**3436,63**	1696,81	1682,22
	Memória	854,09	857,73	847,89	857,98	855,34	826,50	828,33
1000000	Tempo	-	-	**3324,76**	-	-	2317,7	2208,66
	Memória	-	-	1119,94	-	-	1101,28	1101,42
Número de pontos	Resultados	*ult_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	114,81	111,72	90,58	119,39	184,08	75,17	70,99
	Memória	23,61	23,44	23,97	23,45	25,45	25,65	25,82
50000	Tempo	292,28	278,08	229,21	320,10	528,92	178,25	173,73
	Memória	44,13	46,53	45,63	44,30	44,28	43,66	43,70
75000	Tempo	509,92	479,07	392,62	573,94	963,90	298,74	293,30
	Memória	66,09	63,47	64,75	65,57	66,01	61,80	64,54
100000	Tempo	760,37	703,32	584,41	876,79	1509,40	432,98	437,41
	Memória	85,152	82,97	84,19	85,45	87,94	82,53	82,23
250000	Tempo	2761,46	2591,84	2054,44	3521,22	-	1319,06	1392,17
	Memória	202,06	202,75	202,82	203,05	-	200,62	200,23
500000	Tempo	-	-	-	-	-	2761,61	2862,75
	Memória	-	-	-	-	-	397,05	398,80
750000	Tempo	-	-	-	-	-	-	-
	Memória	-	-	-	-	-	-	-
1000000	Tempo	-	-	-	-	-	-	-
	Memória	-	-	-	-	-	-	-
Número de pontos	Resultados	*tet_inc*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	**41,21**	**41,15**	**37,52**	**42,03**	**55,49**	**32,87**	**31,02**
	Memória	25,80	26,14	24,94	23,44	25,39	23,71	25,17
50000	Tempo	**91,76**	**88,01**	**84,58**	**97,32**	**138,49**	**72,20**	**68,85**
	Memória	44,89	43,78	45,02	45,83	44,50	43,92	42,75
75000	Tempo	**148,88**	**142,23**	**133,21**	**161,68**	252,91	**111,68**	**108,38**
	Memória	65,78	62,18	63,00	64,24	63,71	61,30	62,25
100000	Tempo	**206,03**	**198,90**	**185,35**	**235,19**	390,06	**154,42**	**152,90**
	Memória	82,51	85,02	83,69	84,52	85,53	82,54	80,71
250000	Tempo	**628,25**	**598,22**	**563,74**	856,14	1735,93	**438,32**	**431,48**
	Memória	201,98	199,74	202,85	201,42	203,73	195,34	197,73
500000	Tempo	**1571,76**	**1470,01**	**1353,79**	2683,50	-	**939,37**	**928,11**
	Memória	396,53	397,62	397,69	398,48	-	395,03	395,06
750000	Tempo	**2734,64**	2624,56	**2291,60**	-	-	**1435,21**	**1451,45**
	Memória	594,836	592,74	595,03	-	-	589,59	590,44
1000000	Tempo	-	-	3418,17	-	-	**1974,63**	**1940,99**
	Memória	-	-	791,08	-	-	788,15	790,57

Nas Tabelas entre 21 e 27 são apresentados as médias de custo computacional, o coeficiente de variação, o desvio padrão e as ocupações de memória obtidos nas 7 distribuições de pontos, para as 6 ordens de inserção de pontos. Nas Figuras 38, 39 e 40 são apresentadas as médias de custo computacional e as ocupações de memória para as 6 ordens

Tabela 19 Custo computacional (em segundos) e ocupação de memória (em *megabytes*) do algoritmo incremental utilizando a ordem do percurso em largura na árvore rubro-negra, com as 3 abordagens para a busca do tetraedro que contém o novo ponto inserido, nas 7 distribuições de pontos.

Inserção na ordem do percurso em largura na árvore rubro-negra								
Número de pontos	Resultados	*hist_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	1890,01	1838,88	1799,00	**1556,13**	1701,20	2402,17	1548,53
	Memória	29,50	26,87	31,58	26,73	28,83	32,99	33,88
Número de pontos	Resultados	*ult_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	-	-	-	3555,00	-	-	-
	Memória	-	-	-	28,22	-	-	-
Número de pontos	Resultados	*tet_inc*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	-	-	-	3468,74	-	-	-
	Memória -	-	-	29,07	-	-	-	

Tabela 20 Custo computacional (em segundos) e ocupação de memória (em *megabytes*) do algoritmo utilizando a inserção de pontos aleatória, com as 3 abordagens para a busca do tetraedro que contém o ponto inserido, nas 7 distribuições de pontos.

Inserção de pontos aleatória								
Número de pontos	Resultados	*hist_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	-	-	-	**1529,63**	1540,38	-	-
	Memória	-	-	-	26,43	23,82	-	-
Número de pontos	Resultados	*ult_tet*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	-	-	-	-	-	-	-
	Memória	-	-	-	-	-	-	-
Número de pontos	Resultados	*tet_inc*						
		Aleatória	Cilindro	Disco	Planos	Eixos	Parabolóide	Espiral
25000	Tempo	-	-	-	3444,63	-	-	-
	Memória	-	-	-	30,31	-	-	-

de inserção de pontos, na distribuição aleatória, em cilindro e em disco, respectivamente. Nas Figuras 41, 42, 43 e 44 são apresentadas as médias de custo computacional e as ocupações de memória para as 6 ordens de inserção de pontos, na distribuição em torno dos planos, em torno dos eixos, em parabolóide e em espiral, respectivamente.

5.4 Análise dos resultados

Os testes realizados para estruturas tridimensionais também ocorreram somente para instâncias com até 1000000 de pontos. Isso

Tabela 21 Média de custo computacional (em segundos), desvio padrão (σ), coeficiente de variação (CV) e ocupação de memória (em *megabytes*) para os 6 algoritmos na distribuição aleatória. O símbolo - indica que a execução foi abortada após 60 minutos. Máquina utilizada: M2.

Número de pontos	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert (*tet_inc*)	Curva de Lebesgue (*hist_tet*)	*H-indexing* (*tet_inc*)	Espiral (*tet_inc*)	Rubro-negra em ordem (*tet_inc*)
25000	Tempo	**16,01**	21,29	29,18	41,49	41,26	41,29
	σ	0,09	0,15	0,30	0,13	0,11	0,09
	CV(%)	0,56	0,71	1,03	0,32	0,28	0,23
	Memória	26,73	32,83	40,52	27,15	**24,98**	25,80
50000	Tempo	**32,29**	43,61	61,69	91,01	91,14	91,61
	σ	0,07	0,22	0,65	0,20	0,28	0,42
	CV(%)	0,22	0,51	1,06	0,22	0,31	0,46
	Memória	49,01	55,43	57,84	48,62	49,40	**44,89**
75000	Tempo	**49,14**	66,62	94,49	144,01	143,91	147,18
	σ	0,64	0,70	1,13	0,44	0,70	1,07
	CV(%)	1,31	1,05	1,20	0,30	0,49	0,73
	Memória	69,62	74,53	83,44	69,69	67,44	**65,78**
100000	Tempo	**65,17**	89,30	128,24	199,71	199,96	206,43
	σ	0,31	0,86	1,71	0,45	0,56	1,31
	CV(%)	0,48	0,97	1,33	0,22	0,28	0,63
	Memória	93,12	98,78	113,33	96,23	89,30	**82,51**
250000	Tempo	**163,06**	228,01	343,58	581,25	580,79	628,31
	σ	0,09	0,43	2,34	3,18	5,64	0,08
	CV(%)	0,05	0,18	0,68	0,54	0,97	0,01
	Memória	217,49	222,75	255,76	220,36	214,27	**201,98**
500000	Tempo	**325,07**	461,47	711,88	1364,16	1348,61	1558,35
	σ	2,49	3,02	5,28	2,81	3,46	12,10
	CV(%)	0,76	0,65	0,74	0,20	0,25	0,77
	Memória	434,68	438,82	488,63	433,87	429,25	**396,53**
750000	Tempo	**493,28**	696,95	1096,91	2280,52	2278,42	2734,87
	σ	3,73	0,82	18,91	4,35	4,13	4,80
	CV(%)	0,75	0,11	1,72	0,19	0,18	0,17
	Memória	665,76	660,46	720,46	644,29	644,04	**594,83**
1000000	Tempo	**653,40**	932,48	1509,45	3344,48	3339,69	-
	σ	4,18	2,58	31,16	50,03	8,11	-
	CV(%)	0,64	0,27	2,06	1,49	0,24	-
	Memória	863,04	887,34	943,60	865,55	**854,19**	-
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	0	0	0	0	2	**6**

porque, assim como em estruturas bidimensionais, a inserção de pontos na tetraedrização de Delaunay pela ordem da *cut-longest-edge kd-tree* apresentou os melhores custos computacionais, para todas as distribuições de pontos, em todas as instâncias. A inserção de pontos pela ordem da curva de Hilbert apresentou o segundo melhor custo computacional, em todas as distribuições de pontos, e, em seguida, a inserção de pontos pela ordem da curva de Lebesgue.

Dentre as três abordagens para a busca do novo ponto inserido, a

Tabela 22 Média de custo computacional (em segundos), desvio padrão (σ), coeficiente de variação (CV) e ocupação de memória (em *megabytes*) para os 6 algoritmos incrementais na distribuição em cilindro. O símbolo - indica que a execução foi abortada após 60 minutos. Máquina utilizada: M1.

Número de pontos	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert (*tet_inc*)	Curva de Lebesgue (*hist_tet*)	*H-indexing* (*tet_inc*)	Espiral (*tet_inc*)	Rubro-negra em ordem (*hist_tet*)
25000	Tempo	**15,60**	20,87	28,12	39,96	40,10	48,55
	σ	0,37	0,30	0,36	0,62	0,32	0,40
	CV(%)	2,42	1,47	1,30	1,57	0,81	0,82
	Memória	27,97	33,43	37,15	**24,34**	27,58	35,16
50000	Tempo	**31,35**	42,86	58,38	88,61	87,29	108,01
	σ	0,71	0,24	0,23	0,99	0,89	1,16
	CV(%)	2,28	0,57	0,40	1,11	1,02	1,08
	Memória	**48,98**	54,07	56,88	51,15	49,75	60,69
75000	Tempo	**46,76**	64,71	92,22	140,97	138,95	171,82
	σ	0,44	0,70	1,37	2,07	1,42	0,80
	CV(%)	0,95	1,09	1,49	1,46	1,02	0,46
	Memória	**68,74**	75,50	83,91	70,80	71,49	89,16
100000	Tempo	**65,16**	87,84	130,38	193,98	192,56	239,13
	σ	0,95	2,32	2,84	2,99	1,51	0,70
	CV(%)	1,45	2,64	2,18	1,54	0,78	0,29
	Memória	94,44	95,20	105,38	**89,42**	87,83	115,18
250000	Tempo	**156,46**	223,70	328,66	556,92	562,19	697,91
	σ	1,00	2,40	1,27	15,37	12,88	5,17
	CV(%)	0,63	1,07	0,38	2,76	2,29	0,74
	Memória	220,59	222,14	242,97	**216,48**	217,93	282,94
500000	Tempo	**316,77**	459,84	688,93	1284,42	1272,82	1594,35
	σ	8,17	5,50	8,40	20,48	12,05	9,26
	CV(%)	2,57	1,19	1,22	1,59	0,94	0,58
	Memória	434,58	446,07	474,74	**431,97**	441,11	570,34
750000	Tempo	**482,14**	684,78	1052,50	2109,78	2123,48	2623,29
	σ	10,86	2,74	5,63	20,48	8,60	10,26
	CV(%)	2,25	0,40	0,53	0,97	0,40	0,39
	Memória	667,66	666,99	718,96	642,23	**642,04**	857,73
1000000	Tempo	**634,71**	913,30	1468,27	3046,79	3074,40	-
	σ	17,10	12,11	34,41	24,61	27,59	-
	CV(%)	2,69	1,32	2,34	0,80	0,89	-
	Memória	864,89	895,78	960,59	861,63	**857,60**	-
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	2	0	0	**4**	2	0

curva de Hilbert apresentou os melhores custos computacionais ao utilizar a abordagem *tet_inc*, enquanto a curva de Lebesgue apresentou melhores custos computacionais ao se utilizar a abordagem *hist_tet*. Nas demais ordens de inserção de pontos, os melhores custos computacionais alternaram entre as abordagens *tet_inc* e *hist_tet*. A abordagem *ult_tet*, utilizada para a busca do novo ponto inserido na tetraedrização, não apresentou custos computacionais competitivos com os custos computacionais obtidos ao se

Tabela 23 Média de custo computacional (em segundos), desvio padrão (σ), coeficiente de variação (CV) e ocupação de memória (em *megabytes*) para os 6 algoritmos na distribuição em disco. O símbolo - indica que a execução foi abortada após 60 minutos. Máquina utilizada: M3.

Número de pontos	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert (*tet_inc*)	Curva de Lebesgue (*hist_tet*)	*H-indexing* (*hist_tet*)	Espiral (*hist_tet*)	Rubro-negra em ordem (*hist_tet*)
25000	Tempo	**16,77**	22,16	29,26	72,15	70,33	46,07
	σ	0,04	0,10	0,34	0,39	0,24	0,16
	CV(%)	0,28	0,46	1,19	0,54	0,34	0,35
	Memória	**28,05**	33,42	35,98	35,07	36,14	30,78
50000	Tempo	**33,88**	45,98	61,64	160,58	159,03	102,69
	σ	0,12	0,69	0,34	0,64	0,99	0,63
	CV(%)	0,38	1,50	0,55	0,40	0,62	0,61
	Memória	**48,67**	55,55	59,33	64,82	69,15	59,62
75000	Tempo	**51,45**	69,20	95,78	261,36	255,80	165,29
	σ	0,31	0,37	0,53	2,72	1,79	1,30
	CV(%)	0,60	0,54	0,55	1,04	0,70	0,78
	Memória	**68,25**	78,17	86,45	96,43	96,85	86,63
100000	Tempo	**68,39**	93,26	130,58	365,97	364,47	228,21
	σ	0,47	0,65	0,57	2,21	3,13	0,68
	CV(%)	0,70	0,70	0,44	0,60	0,86	0,29
	Memória	**93,50**	99,25	111,33	129,83	124,04	116,57
250000	Tempo	**171,42**	239,64	350,99	1103,75	1107,43	665,87
	σ	0,65	0,62	3,08	4,40	10,58	3,01
	CV(%)	0,38	0,25	0,87	0,39	0,95	0,45
	Memória	**220,04**	249,33	263,92	305,11	308,06	280,25
500000	Tempo	**345,08**	488,56	734,80	2539,82	2601,26	1480,08
	σ	2,32	3,74	4,16	21,57	26,48	5,11
	CV(%)	0,67	0,76	0,56	0,84	1,01	0,34
	Memória	**432,72**	503,02	538,93	613,43	607,53	556,55
750000	Tempo	**525,97**	745,32	1140,04	-	-	2431,35
	σ	9,89	14,68	21,95	-	-	93,83
	CV(%)	1,88	1,96	1,92	-	-	3,85
	Memória	**665,69**	768,66	823,42	-	-	847,89
1000000	Tempo	**692,32**	1003,49	1562,57	-	-	3322,28
	σ	4,01	5,83	13,26	-	-	14,26
	CV(%)	0,57	0,58	0,84	-	-	0,42
	Memória	**861,19**	1038,47	1100,35	-	-	1119,94
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	**8**	0	0	0	0	0

utilizar as abordagens *hist_tet* e *tet_inc*.

Alguns algoritmos incrementais (com os esquemas de inserção de pontos de acordo com o percurso em ordem na árvore rubro-negra e com o esquema de indexação *H-indexing*) utilizaram menor ocupação de memória que o algoritmo incremental utilizando a ordem da *cut-longest-edge kd-tree* em 4 distribuições de pontos, que foram: distribuição aleatória, pontos em um cilindro, pontos em um parabolóide e pontos em um espiral. Porém, a ocupação de memória do algoritmo incremental de Liu, Yan e Lo (2013) foi

Tabela 24 Média de custo computacional (em segundos), desvio padrão (σ), coeficiente de variação (CV) e ocupação de memória (em *megabytes*) para os 6 algoritmos na distribuição em torno dos planos. O símbolo - indica que a execução foi abortada após 60 minutos. Máquina utilizada para testes: M1.

Número de pontos	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert (*tet_inc*)	Curva de Lebesgue (*hist_tet*)	*H-indexing* (*hist_tet*)	Espiral (*hist_tet*)	Rubro-negra em ordem (*hist_tet*)
25000	Tempo	**15,62**	20,48	25,87	51,95	51,87	50,81
	σ	0,26	0,21	0,21	1,15	0,51	0,21
	CV(%)	1,66	1,05	0,82	2,21	0,98	0,42
	Memória	**27,00**	32,76	35,25	36,30	33,48	33,34
50000	Tempo	**31,67**	41,88	54,79	115,63	115,60	115,05
	σ	0,71	0,16	0,61	1,83	1,22	0,76
	CV(%)	2,25	0,38	1,12	1,58	1,06	0,66
	Memória	**48,40**	54,65	58,94	65,25	63,40	60,46
75000	Tempo	**47,71**	64,07	83,72	184,27	185,89	184,80
	σ	0,60	0,74	0,75	1,18	2,55	0,95
	CV(%)	1,27	1,15	0,90	0,64	1,37	0,51
	Memória	**68,66**	78,20	83,85	92,88	94,17	90,41
100000	Tempo	**62,54**	86,24	114,58	256,77	262,91	260,95
	σ	0,49	1,63	1,54	0,90	2,56	5,40
	CV(%)	0,78	1,89	1,34	0,35	0,97	2,06
	Memória	**94,13**	96,94	106,58	127,69	122,02	115,23
250000	Tempo	**155,77**	219,98	310,30	755,93	759,51	791,89
	σ	2,69	0,71	5,77	5,36	7,75	11,03
	CV(%)	1,72	0,32	1,86	0,71	1,02	1,39
	Memória	**218,98**	237,69	252,56	298,66	301,78	286,41
500000	Tempo	**323,64**	447,43	645,74	1723,72	1718,67	1793,14
	σ	3,88	4,75	5,40	8,37	1,12	7,45
	CV(%)	1,20	1,06	0,83	0,48	0,06	0,41
	Memória	**435,03**	487,96	517,14	597,52	599,75	566,37
750000	Tempo	**474,46**	677,80	981,92	2780,08	2809,49	3030,32
	σ	3,35	9,40	5,71	63,14	22,26	60,40
	CV(%)	0,70	1,38	0,58	2,27	0,79	1,99
	Memória	**663,95**	741,77	796,64	913,16	910,23	857,98
1000000	Tempo	**643,03**	905,81	1343,92	-	-	-
	σ	10,58	11,71	12,65	-	-	-
	CV(%)	1,64	1,29	0,94	-	-	-
	Memória	**859,52**	1007,83	1059,63	-	-	-
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	**8**	0	0	0	0	0

competitiva com a ocupação de memória desses algoritmos. É possível que em uma outra implementação, o algoritmo de Liu, Yan e Lo (2013) utilize menor ocupação de memória que a implementação utilizada neste trabalho.

Dentre todas as ordens de inserção de pontos testadas, a inserção aleatória foi a que, em geral, apresentou piores custos computacionais para a computação da tetraedrização de Delaunay. A inserção aleatória de pontos obteve melhor custo computacional apenas na distribuição em torno dos

Tabela 25 Média de custo computacional (em segundos), desvio padrão (σ), coeficiente de variação (CV) e ocupação de memória (em *megabytes*) para os algoritmos incrementais nem torno dos eixos. O símbolo - indica que a execução foi abortada após 60 minutos. Máquina utilizada: M2.

Número de pontos	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert (*tet_inc*)	Curva de Lebesgue (*hist_tet*)	*H-indexing* (*tet_inc*)	Espiral (*hist_tet*)	Rubro-negra em ordem (*hist_tet*)
25000	Tempo	**19,70**	20,47	25,40	50,56	58,98	61,07
	σ	0,08	0,14	0,07	0,40	0,51	0,41
	CV(%)	0,42	0,70	0,31	0,80	0,87	0,68
	Memória	27,57	37,21	40,24	**25,34**	35,42	37,01
50000	Tempo	**36,19**	42,65	55,31	117,17	131,15	138,41
	σ	0,43	0,24	0,43	0,71	1,55	2,44
	CV(%)	1,20	0,57	0,77	0,61	1,18	1,76
	Memória	**49,74**	66,45	68,87	50,33	64,48	61,57
75000	Tempo	**50,97**	66,01	86,37	197,24	212,52	221,75
	σ	0,58	0,35	0,39	1,22	2,96	1,20
	CV(%)	1,14	0,53	0,45	0,62	1,39	0,54
	Memória	**68,22**	91,39	85,08	72,38	94,94	89,37
100000	Tempo	**69,06**	87,52	117,42	288,69	295,07	313,10
	σ	0,38	0,48	1,03	1,14	2,08	3,06
	CV(%)	0,55	0,55	0,88	0,39	0,70	0,97
	Memória	**93,96**	118,03	125,05	98,45	125,37	115,84
250000	Tempo	**208,41**	227,62	318,93	858,75	881,29	942,53
	σ	0,30	0,93	0,39	15,19	8,70	8,20
	CV(%)	0,14	0,41	0,12	1,76	0,98	0,87
	Memória	234,78	260,52	278,13	**224,51**	304,55	285,31
500000	Tempo	**335,06**	490,29	714,34	1535,76	1996,23	2169,20
	σ	1,21	1,41	2,36	6,11	17,09	13,39
	CV(%)	0,36	0,28	0,33	0,39	0,85	0,61
	Memória	**431,70**	490,46	523,22	447,18	601,17	568,66
750000	Tempo	**517,32**	754,11	1104,10	2106,14	3269,56	3489,51
	σ	4,05	2,33	5,17	3,50	30,20	45,91
	CV(%)	0,78	0,30	0,46	0,16	0,92	1,31
	Memória	668,12	710,39	765,28	**655,30**	925,30	855,34
1000000	Tempo	**864,39**	1175,34	1656,43	2739,21	-	-
	σ	1,52	11,18	5,51	16,52	-	-
	CV(%)	0,17	0,95	0,33	0,60	-	-
	Memória	936,37	931,39	992,28	**876,79**	-	-
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	**4**	0	0	**4**	0	0

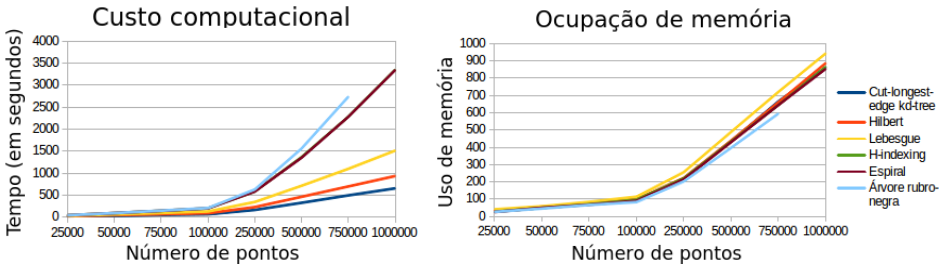
planos e em torno dos eixos quando foi utilizada a inserção dos pontos de acordo com a ordem da árvore rubro-negra utilizando o percurso em largura. Em geral, o algoritmo incremental utilizando a inserção de pontos de acordo com o percurso em largura na árvore rubro-negra foi o que apresentou o segundo pior custo computacional.

Tabela 26 Média de custo computacional (em segundos), desvio padrão (σ), coeficiente de variação (CV) e ocupação de memória (em *megabytes*) para os 6 algoritmos na distribuição em parabolóide. Máquina utilizada: M3.

Número de pontos	Resultados	*cut-longest edge kd-tree*	Curva de Hilbert (*tet_inc*)	Curva de Lebesgue (*hist_tet*)	*H-indexing* (*tet_inc*)	Espiral (*hist_tet*)	Rubro-negra em ordem (*tet_inc*)
25000	Tempo	**15,51**	22,56	29,33	34,74	42,84	32,92
	σ	0,06	0,13	0,15	0,09	0,27	0,22
	CV(%)	0,39	0,61	0,52	0,26	0,64	0,69
	Memória	24,83	34,37	37,84	28,75	34,90	**23,71**
50000	Tempo	**31,53**	46,49	61,04	75,62	93,41	71,61
	σ	0,10	0,40	0,41	0,12	0,44	0,35
	CV(%)	0,33	0,86	0,67	0,16	0,47	0,50
	Memória	47,91	59,52	63,57	46,23	57,78	**43,92**
75000	Tempo	**48,28**	71,56	93,31	119,50	148,13	111,83
	σ	0,09	0,42	1,00	0,76	0,66	0,45
	CV(%)	0,20	0,59	1,08	0,63	0,45	0,40
	Memória	66,5	86,56	95,15	74,00	93,76	**61,30**
100000	Tempo	**64,62**	95,19	125,71	165,50	204,99	154,54
	σ	0,39	0,23	0,98	0,78	1,50	0,86
	CV(%)	0,61	0,24	0,78	0,47	0,73	0,55
	Memória	90,92	111,69	122,16	89,53	116,10	**82,54**
250000	Tempo	**163,43**	247,88	339,74	466,04	582,64	435,97
	σ	0,13	3,33	3,99	0,82	7,57	2,09
	CV(%)	0,08	1,34	1,17	0,17	1,30	0,48
	Memória	210,90	280,50	286,61	231,02	284,67	**195,34**
500000	Tempo	**332,78**	513,51	711,63	1013,35	1337,75	939,19
	σ	2,72	5,18	1,86	5,12	13,98	6,99
	CV(%)	0,82	1,00	0,26	0,50	1,04	0,74
	Memória	422,27	537,74	568,26	462,44	574,59	**395,03**
750000	Tempo	**502,74**	810,99	1189,73	1601,58	2198,20	1445,48
	σ	2,72	12,65	1,51	9,82	16,95	9,95
	CV(%)	0,54	1,56	0,12	0,61	0,77	0,68
	Memória	648,95	788,91	824,625	679,82	876,56	**589,59**
1000000	Tempo	**667,19**	1072,25	1580,52	2209,74	3108,80	1972,73
	σ	3,04	12,05	1,10	19,02	14,27	1,40
	CV(%)	0,45	1,12	0,06	0,86	0,45	0,07
	Memória	844,01	1038,68	1090,79	924,18	1148,20	**788,15**
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	0	0	0	0	0	**8**

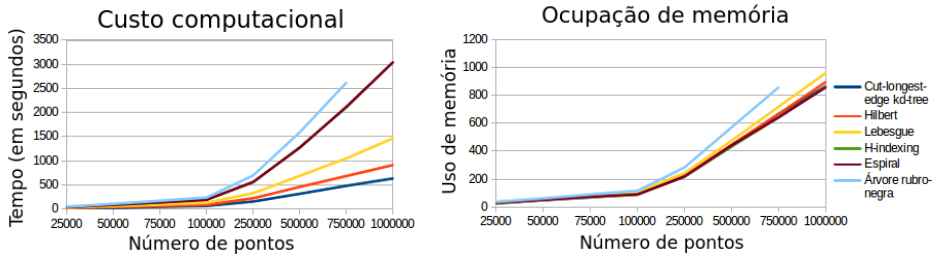
Tabela 27 Média de custo computacional (em segundos), desvio padrão (σ), coeficiente de variação (CV) e ocupação de memória (em *megabytes*) dos 6 algoritmos incrementais na distribuição espiral. Máquina utilizada: M1.

Número de pontos	Resultados	*cut-longest-edge kd-tree*	Curva de Hilbert (*tet_inc*)	Curva de Lebesgue (*hist_tet*)	*H-indexing* (*tet_inc*)	Espiral (*hist_tet*)	Rubro-negra em ordem (*tet_inc*)
25000	Tempo	**15,24**	20,01	24,78	33,26	40,31	31,21
	σ	0,25	0,06	0,10	0,25	0,23	0,23
	CV(%)	1,69	0,34	0,43	0,76	0,57	0,76
	Memória	26,09	34,11	37,75	27,18	31,43	**25,17**
50000	Tempo	**31,18**	41,70	51,99	73,08	91,28	68,53
	σ	0,68	0,58	1,06	0,97	1,68	0,47
	CV(%)	2,19	1,41	2,05	1,33	1,84	0,69
	Memória	48,14	58,47	63,35	46,46	60,34	**42,75**
75000	Tempo	**46,05**	62,44	79,80	115,88	144,09	114,41
	σ	0,58	0,47	0,71	1,18	4,54	4,65
	CV(%)	1,28	0,75	0,89	1,02	3,15	4,07
	Memória	68,65	86,79	91,41	70,05	89,38	**62,25**
100000	Tempo	**61,75**	83,33	106,94	161,01	198,42	151,42
	σ	0,43	0,54	3,73	2,85	3,69	1,12
	CV(%)	0,70	0,65	3,49	1,77	1,86	0,74
	Memória	91,37	110,25	117,81	92,41	119,42	**80,71**
250000	Tempo	**155,84**	216,30	296,21	459,36	577,57	447,89
	σ	4,23	0,80	4,97	12,08	18,20	17,99
	CV(%)	2,71	0,37	1,67	2,62	3,15	4,01
	Memória	216,26	267,91	283,44	230,15	289,03	**197,73**
500000	Tempo	**311,63**	440,59	614,88	980,84	1285,08	937,39
	σ	3,71	6,88	0,75	4,07	1,38	10,36
	CV(%)	1,19	1,56	0,12	0,41	0,10	1,10
	Memória	428,87	532,32	560,37	436,21	577,99	**395,06**
750000	Tempo	**466,85**	670,91	953,25	1552,99	2098,09	1464,37
	σ	4,47	14,71	8,45	35,67	42,89	11,74
	CV(%)	0,95	2,19	0,88	2,29	2,04	0,80
	Memória	658,70	776,41	830,92	670,73	886,57	**590,44**
1000000	Tempo	**619,16**	975,30	1398,54	2129,26	2869,00	1921,53
	σ	6,82	14,82	44,89	11,45	69,15	18,25
	CV(%)	1,10	1,52	3,21	0,53	2,41	0,94
	Memória	855,36	1026,84	1075,11	898,90	1156,00	**790,57**
Número de resultados melhores	Tempo	**8**	0	0	0	0	0
	Memória	0	0	0	0	0	**8**



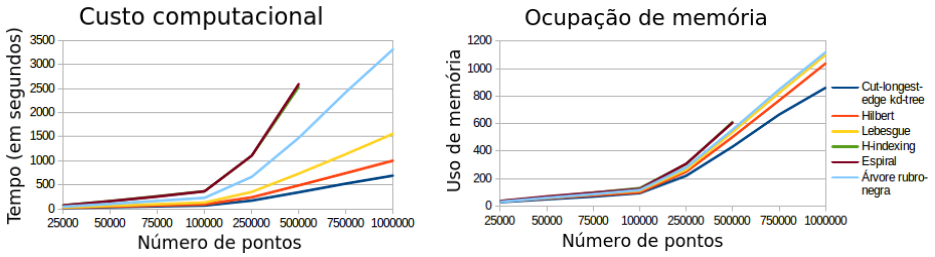
(a) Custo computacional. (b) Ocupação de memória.

Figura 38 Custos computacionais e ocupações de memória dos 6 algoritmos incrementais em conjuntos de pontos distribuídos de maneira aleatória.



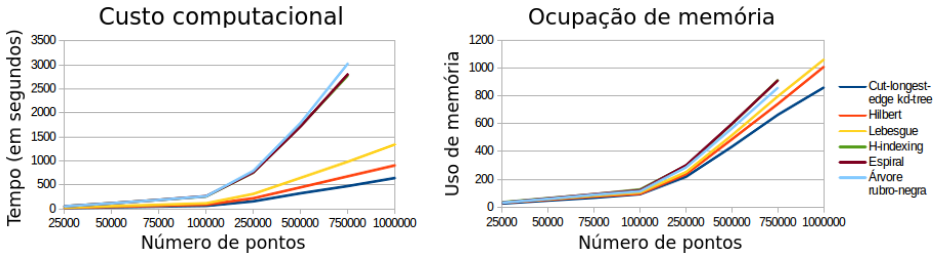
(a) Custo computacional. (b) Ocupação de memória.

Figura 39 Custos computacionais e as ocupações de memória dos 6 algoritmos incrementais em conjuntos de pontos distribuídos em cilindro.



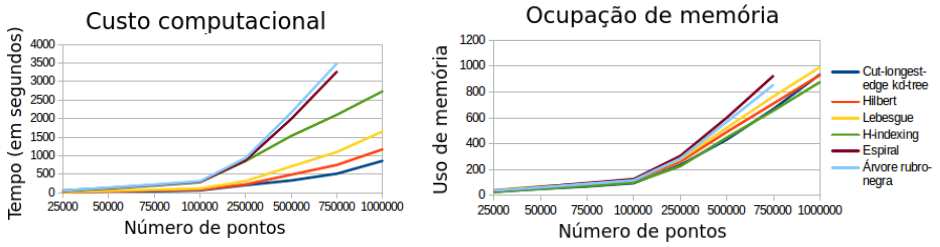
(a) Custo computacional. (b) Ocupação de memória.

Figura 40 Custos computacionais e ocupações de memória dos 6 algoritmos incrementais em conjuntos de pontos distribuídos em disco.



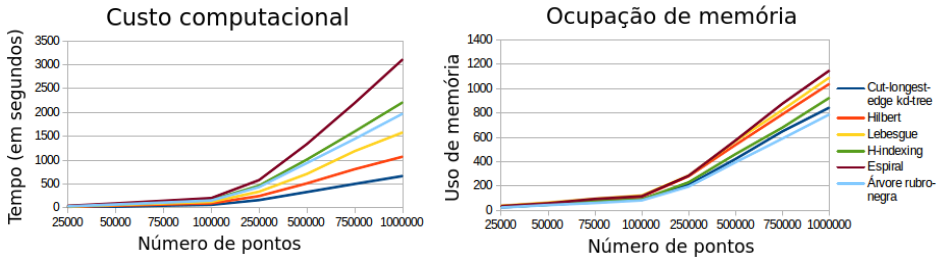
(a) Custo computacional. (b) Ocupação de memória.

Figura 41 Custos computacionais e ocupações de memória dos 6 algoritmos incrementais em conjuntos de pontos distribuídos em torno dos planos.



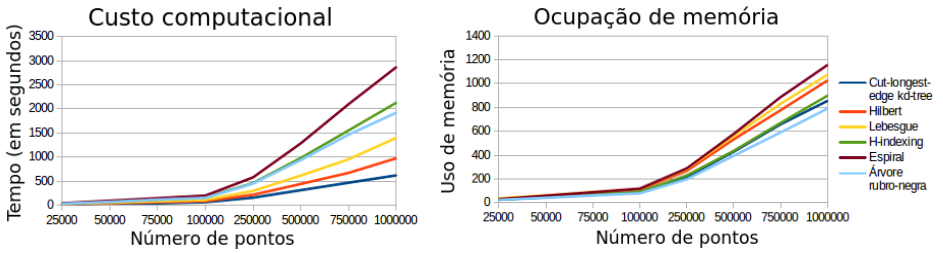
(a) Custo computacional. (b) Ocupação de memória.

Figura 42 Custos computacionais e ocupações de memória dos 6 algoritmos incrementais em conjuntos de pontos distribuídos em torno dos eixos.



(a) Custo computacional. (b) Ocupação de memória.

Figura 43 Custos computacionais e ocupações de memória dos 6 algoritmos incrementais em conjuntos de pontos distribuídos em parabolóide.



(a) Custo computacional. (b) Ocupação de memória.

Figura 44 Custos computacionais e ocupações de memória dos 6 algoritmos incrementais em conjuntos de pontos distribuídos em espiral.

6 CONCLUSÕES E TRABALHOS FUTUROS

Quatro sequências de inserção de pontos para algoritmos incrementais para a geração da tesselação de Delaunay foram propostas: a ordem da árvore rubro-negra, com percursos em ordem e em largura, a ordem espiral e a ordem do *H-indexing*. Os custos computacionais e ocupações de memória desses algoritmos foram comparados com os custos computacionais e ocupações de memória dos algoritmos incrementais utilizando a ordem da *cut-longest-edge kd-tree*, pelas ordens das curvas de Hilbert e de Lebesgue e pela inserção de pontos aleatória.

Os testes foram realizados em estruturas bidimensionais e em estruturas tridimensionais no intervalo unitário. Em estruturas bidimensionais, realizaram-se testes em 6 distribuições de pontos e em estruturas tridimensionais, os testes ocorreram em 7 distribuições de pontos.

Os testes foram realizados em conjuntos com até 1000000 de pontos. Não foram realizados testes em instâncias maiores porque o algoritmo incremental com a inserção de pontos na ordem da *cut-longest-edge kd-tree* apresentou o melhor custo computacional, em todas as distribuições de pontos, em todas as instâncias. Em todos os testes, utilizou-se a biblioteca de precisão MPFR para a realização dos testes de orientação, de circuncírculo e de circunsfera.

Os resultados obtidos neste trabalho corroboram com os testes realizados por Liu, Yan e Lo (2013), que mostram que o custo computacional do algoritmo incremental utilizando a inserção de pontos na ordem da curva de Hilbert foi superior ao custo apresentado ao utilizar a ordem da *cut-*

longest-edge kd-tree. Embora os testes de Liu, Yan e Lo (2013) tenham ocorrido em estruturas tridimensionais, mostrou-se neste trabalho que o custo computacional ao se utilizar a ordem dada pelo percurso em largura na *cut-longest-edge kd-tree* em estruturas bidimensionais também é inferior ao custo computacional do algoritmo incremental utilizando a ordem da curva de Hilbert para a inserção de pontos na triangulação de Delaunay.

Em alguns testes, algoritmos incrementais, como, por exemplo, o algoritmo utilizando o percurso em ordem na árvore rubro-negra (em estruturas bidimensionais e tridimensionais) e utilizando a ordem do indexador *H-indexing* (em estruturas tridimensionais) para determinar a ordem em que os pontos são inseridos no domínio, apresentaram uma ocupação de memória inferior à memória utilizada pelo algoritmo incremental utilizando a ordem da *cut-longest-edge kd-tree*. Porém, a ocupação de memória do algoritmo incremental de Liu, Yan e Lo (2013) foi competitiva com todos esses algoritmos, em todos os testes. É possível que uma outra implementação do algoritmo incremental utilizando a ordem da *cut-longest-edge kd-tree* possa apresentar uma ocupação de memória menor que a implementação utilizada neste trabalho.

Ao utilizar o algoritmo incremental de Liu, Yan e Lo (2013), o domínio é subdividido de maneira mais uniforme que os demais algoritmos incrementais implementados neste trabalho. É provável que, por esse motivo, seja necessário um número menor de testes de orientação, testes do circuncírculo (em estruturas bidimensionais) e testes da circunsfera (para estruturas tridimensionais) no algoritmo incremental de Liu, Yan e Lo (2013) quando comparado ao número de testes realizados nos demais algoritmos incrementais avaliados neste trabalho. Além disso, ao utilizar

o algoritmo de Liu, Yan e Lo (2013), um número menor de poliedros são criados e destruídos quando comparado ao número de criações e deleções de poliedros nos outros 7 algoritmos avaliados. Tanto em estruturas bidimensionais quanto em estruturas tridimensionais, os custos computacionais obtidos ao se utilizar a inserção aleatória de pontos e a inserção de pontos de acordo com o percurso em largura na árvore rubro-negra não foram competitivos com os demais algoritmos implementados.

O algoritmo incremental de Liu, Yan e Lo (2013) é o possível estado da arte para a geração da tetraedrização de Delaunay. Como esse algoritmo foi superado pelo algoritmo incremental de Lo (2013) em estruturas bidimensionais, o algoritmo de Lo (2013) é o possível estado da arte para a geração da triangulação de Delaunay.

Trabalhos futuros referem-se à realização da paralelização desses algoritmos incrementais, tanto em estruturas bidimensionais quanto em estruturas tridimensionais. Pretende-se avaliar o custo computacional e a ocupação de memória desses algoritmos, comparando os resultados obtidos após a paralelização com os resultados obtidos utilizando a implementação sequencial.

Além disso, como trabalhos futuros, pretende-se implementar as variações dos testes de orientação (para estruturas bidimensionais e tridimensionais), de circuncírculo (para estruturas bidimensionais) e de circunsfera (para estruturas bidimensionais) propostas em Shewchuk (1997a). Será verificado se a utilização dessas variações de testes diminuirão os custos computacionais dos algoritmos incrementais implementados neste trabalho.

REFERÊNCIAS

AMENTA, N.; CHOI, S.; ROTE, G. Incremental constructions con BRIO. In: ANNUAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY, 9., 2003, New York. **Proceedings...** New York: ACM, 2003. p. 211–219.

ANTONY, W. B. A.; VIGILA, S. M. C. Biometric encryption system based on Delaunay indexing method. In: INTERNATIONAL CONFERENCE ON CIRCUITS, POWER AND COMPUTING TECHNOLOGIES, 1., 2013, Nagercoil. **Proceedings...** Nagercoil: IEEE, 2013. p. 1156–1161.

BADER, M. **Space-Filling Curves**: an introduction with applications in scientific computing. Berlin: Springer, 2013. 278 p.

BARBER, C. B.; DOBKIN, D. P.; HUHDANPA, H. The quickhull algorithm for convex hulls. **ACM Transactions on Mathematical Software**, New York, v. 22, n. 4, p. 469–483, Dec. 1996.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. **Communications of the ACM**, New York, v. 18, n. 9, p. 509–517, Sept. 1975.

BOISSONNAT, J. D. et al. Triangulations in CGAL. **Computational Geometry**, Hong Kong, v. 22, n. 1/3, p. 5–19, June 2002.

BOWYER, A. Computing Dirichlet tessellations. **The Computer Journal**, Oxford, v. 24, n. 2, p. 162–166, May 1981.

CANTOR, G. Ein beitrag zur mannigfaltigkeitslehre. **Crelle Journal**, Berlin, v. 84, p. 242–258, 1878.

CHENG, S. W.; DEY, T. K.; SHEWCHUK, J. R. **Delaunay Mesh Generation**. Boca Raton: CRC, 2013. 394 p. (CRC Computer and Information Science Series).

CLARKSON, K. L. Safe and effective determinant evaluation. In: IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 31., 1992, Washington. **Proceedings...** Washington: IEEE Computer Society Press, 1992. p. 387–395.

COMPUTATIONAL GEOMETRY ALGORITHMS LIBRARY. **CGAL user and reference manual**. [S.l.], 2012. 2387 p. Disponível em: <http://doc.cgal.org/latest/Manual/index.html>. Acesso em: 29 jul. 2015.

CONTINUOUS physical distribution node layout optimization method based on weighted Voronoi diagram. CN 102,393,869, Google Patents, 8 maio 2013. Disponível em: <http://www.google.com/patents/CN102393869B?cl=en>. Acesso em: 29 jul. 2015.

DELAUNAY, B. N. Sur la sphere vide. **Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk**, Moscow, v. 7, p. 793–800, 1934.

DEVILLERS, O. Improved incremental randomized Delaunay triangulation. In: ANNUAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY, 14., 1998, New York. **Proceedings...** New York: ACM, 1998. p. 106–115.

DUFF, I. S.; ERISMAN, A. M.; REID, J. K. On George's nested dissection method. **SIAM Journal on Numerical Analysis**, Auckland, v. 13, p. 686–695, Mar. 1976.

DUFF, I. S.; MEURANT, G. A. The effect of ordering on preconditioned conjugate gradients. **BIT**, Lawrence, v. 29, n. 4, p. 635–657, Dec. 1989.

EUGEN NETTO. Beitrag zur Mannigfaltigkeitslchrc. **Crelle Journal**, Berlin, v. 86, p. 263–268, 1879.

FLORIANI, L. D.; KOBBELT, L.; PUPPO, E. A survey on data structures for level-of-detail models. In: DODGSON, N.; FLOATER, M.; SABIN, M. (Ed.). **Advances in Multiresolution for Geometric Modeling and Visualization, Mathematics**. New York: Springer Verlag, 2004. p. 49–74. (Series in Mathematics and Visualization).

FREDERICK, C. O.; WONG, Y. C.; EDGE, F. W. Two-dimensional automatic mesh generation for structural analysis. **International Journal for Numerical Methods in Engineering**, New Jersey, v. 2, n. 1, p. 133–144, Jan./Mar. 1970.

GARDNER, M. In which "monster"curves force redefinition of the word "curve". **Scientific American**, Washington, v. 235, p. 124–133, Dec. 1976.

GRAY, F. **Pulse code communication**. US Patent 2,632,058, 17 mar. 1953. Disponível em: <http://www.google.com/patents/US2632058>. Acesso em: 29 jul. 2015.

GUIBAS, L. J.; SEDGEWICK, R. A dichromatic framework for balanced trees. In: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 19., 1978, Washington. **Proceedings...** Washington: IEEE Computer Society, 1978. p. 8–21.

GUIBAS, L. J.; STOLFI, J. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. **ACM Transactions on Graphics**, New York, v. 4, n. 2, p. 74–123, Apr. 1985.

GYVES, O. D.; TOLEDO, L.; RUDOMÍN, I. Proximity queries for crowd simulation using truncated Voronoi diagrams. In: _____. **Proceedings of motion on games**. New York: ACM, 2013. p. 65:87–65:92.

HAVERKORT, H. **An inventory of three-dimensional Hilbert space-filling curves**. Eindhoven: Cornell University Library, 2011. 25 p. External report.

HILBERT, D. Über die stetige abbildung einer linie auf ein flächenstück. **Mathematische Annalen**, Königsberg, v. 38, p. 459–460, Mar. 1891.

LAWSON, C. L. **Software for C^1 Surface Interpolation**. Washington: JPL, 1977. 194 p. (Technical Report JPL Pub, 77-30).

LEBESGUE, H. **Leçons Sur L'Integration et la Recherche des Fonctions Primitives**. Paris: Gauthier-Villars, 1904. 45 p.

LIU, J. F.; YAN, J. H.; LO, S. H. A new insertion sequence for incremental Delaunay triangulation. **Acta Mechanica Sinica**, Beijing, v. 29, n. 1, p. 99–109, Feb. 2013.

LIU, Y.; SNOEYINK, J. A comparison of five implementations of 3D Delaunay Tessellation. **MSRI Publications**, Cambridge, v. 52, p. 439–458, Aug. 2005.

LO, S. H. Delaunay triangulation of non-uniform point distributions by means of multi-grid insertion. **Finite Elements in Analysis and Design**, Amsterdam, v. 63, p. 8–22, Jan. 2013.

NIEDERMEIER, R.; REINHARDT, K.; SANDERS, P. Towards optimal locality in mesh-indexings. **Discrete applied mathematics**, Amsterdam, v. 117, n. 1/3, p. 211–237, Mar. 2002.

NOGUEIRA, J. R. **Uma revisão da triangulação de Delaunay com malhas geradas pelo algoritmo de Green-Sibson**. 2013. 60 p. Monografia (Especialização em Ciência da Computação) - Universidade Federal de Lavras, Lavras, 2013.

NOGUEIRA, J. R.; OLIVEIRA, S. L. G. de. Algoritmos para a geração da triangulação de Delaunay por meio de inserção incremental de pontos. In: CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UNIVERSIDADE FEDERAL DE LAVRAS, 25., 2012, Lavras. **Anais...** Lavras: PRP-UFLA, 2012. Disponível em: <http://www.prp.ufla.br/ciuflasig/generateResumoPDF.php?id=1815>. Acesso em: 10 mar. 2015.

NOGUEIRA, J. R.; OLIVEIRA, S. L. G. de. Introdução sucinta à triangulação de Delaunay e suas propriedades. In: CONGRESSO DE MATEMÁTICA APLICADA E COMPUTACIONAL, 1., 2011, Uberlândia. **Anais...** Uberlândia: SBMAC, 2011a. 1 CD-ROM.

NOGUEIRA, J. R.; OLIVEIRA, S. L. G. de. Introduction to triangular mesh generation and the Delaunay triangulation. In: IBERIAN LATIN-AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING, 32., 2011, Ouro Preto. **Proceedings...** Ouro Preto: ABMEC, 2011b. 1 CD-ROM.

OLIVEIRA, S. L. G. de. **Algoritmos e seus fundamentos**. Lavras: UFLA, 2011. 420 p.

OLIVEIRA, S. L. G. de; NOGUEIRA, J. R.; TAVARES, J. M. R. S. A systematic review of algorithms with linear-time behaviour to generate Delaunay and Voronoi tessellations. **Computer Modeling in Engineering and Sciences**, Duluth, v. 100, n. 1, p. 31–57, Sept. 2014.

PALIWAL, A. et al. An improved segmentation technique based on Delaunay triangulations for breast infiltration/tumor detection from mammograms. **International Journal of Engineering and Technology**, Singapore, v. 5, n. 3, p. 2565–2574, June/July 2013.

PEANO, G. Sur une courbe, qui remplit toute une aire plane. **Mathematische Annalen**, Berlin, v. 36, p. 157–160, 1890.

ROBAINA, D. T. et al. An adaptive graph for volumetric mesh visualization. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE, 2010, Amsterdam. **Proceedings...** Maryland Heights: Elsevier, 2010. v. 1, p. 1741–1749.

SAGAN, H. **Space Filling Curves**. New York: Springer-Verlag, 1994. 194 p.

SCHAMBERGER, S.; WIERUM, J. M. Partitioning finite element meshes using space-filling curves. **Future Generation Computer Systems**, Krasnoyarsk, v. 21, n. 5, p. 759–766, May 2005.

SCHRIJVERS, O.; BOMMEL, F.; BUCHIN, K. Delaunay triangulations on the word RAM: towards a practical worst-case optimal algorithm. In: EUROPEAN WORKSHOP ON COMPUTATIONAL GEOMETRY, 28., 2012, Assisi. **Proceedings...** Assisi: EuroCG, 2012. p. 13–16.

SHEWCHUK, J. R. Adaptive precision floating-point arithmetic and fast robust geometric predicates. **Discrete & Computational Geometry**, Pittsburgh, v. 18, n. 3, p. 305–363, Oct. 1997a.

SHEWCHUK, J. R. **Delaunay Refinement Mesh Generation**. 1997. 215 p. Thesis (Ph.D. in Computer Science) - Carnegie Mellon University, Pennsylvania, 1997b.

SHEWCHUK, J. R. Tetrahedral mesh generation by Delaunay refinement. In: ANNUAL ACM SYMPOSIUM ON COMPUTATIONAL GEOMETRY, 40., 1998, Minnesota. **Proceedings...** Minnesota: ACM, 1998. p. 86–95.

SIERPIŃSKI, W. Sur une nouvelle courbe continue qui remplit toute une aire plane. **Bulletin de l'Academie des Sciences de Cracovie, Serie A**, Cracovie, v. 1, p. 462–478, 1912.

SNOEYINK, J.; LIU, Y. **Tess3**: a program to compute 3D Delaunay tessellations for well-distributed points. Versão enviada pelo professor J. Snoeyink em julho de 2013.

THIRUSITTAMPALAM, K. et al. A novel framework for cellular tracking and mitosis detection in dense phase contrast microscopy images. **IEEE Journal of Biomedical and Health Informatics**, London, v. 17, n. 3, p. 642–653, May 2013.

VORONOI, G. F. Nouvelles applications des parametres continus a la theorie des formes quadratiques. **Journal fur die Reine und Angewandte Mathematik**, Berlin, v. 133, p. 97–178, 1908.

WATSON, D. F. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. **The Computer Journal**, Oxford, v. 24, n. 2, p. 167–172, May 1981.

WIERUM, J. M. **Definition of a new circular space-filling curve**: β Ω-indexing. Paderborn: Paderborn Center for Parallel Computing, 2002. 12 p. (Technical Report TR-001-02).

WOOD, F. Modeling cell layers on complex surfaces using constrained Voronoi diagrams. In: _____**. ACM special interest group on graphics and interactive techniques 2013 posters.** New York: ACM, 2013. p. 86:1.